Journal of Symbolic Computation 47 (2012) 1533–1560

Contents lists available at SciVerse ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Discovering invariants via simple component analysis

Gianluca Amato, Maurizio Parton, Francesca Scozzari

Dipartimento di Scienze, Università degli Studi "G. d'Annunzio" di Chieti-Pescara, viale Pindaro 42, 65127 Pescara, Italy

ARTICLE INFO

Article history:
Received 10 January 2011
Accepted 8 June 2011
Available online 30 December 2011

Keywords:
Static analysis
Abstract interpretation
Simple component analysis
Intervals
Interval arithmetic

ABSTRACT

We propose a new technique combining dynamic and static analysis of programs to find linear invariants. We use a statistical tool, called *simple component analysis*, to analyze partial execution traces of a given program. We get a new coordinate system in the vector space of program variables, which is used to specialize numerical abstract domains. As an application, we instantiate our technique to interval analysis of simple imperative programs and show some experimental evaluations.

1. Introduction

Analyzing programs to prove numerical properties is a very rich research field, which has received much attention. Typical numerical properties range from basic requirements, such as "all the array indexes are within the correct bounds" or "division by zero cannot happen", to complex loop invariants. Moreover, numerical properties may help other kind of analyses, such as termination analyses (Colóon and Sipma, 2001), timing analyses (Gulavani and Gulwani, 2008), shape analyses (Chang and Rival, 2008), string cleanness analyses (Dor et al., 2001) and so on.

Most of the work in this subject uses either the dynamic or the static approach. In the first case, candidate invariants are "guessed" from a (possibly symbolic) execution of the program (Ernst et al., 2001; Gulwani and Necula, 2005). In the latter case, invariants are proved or derived from an approximated, formal model of the program. Both approaches have drawbacks. In the dynamic approach, it is possible to explore only a finite set of bounded, partial execution traces. Thus, we may only find candidate invariants, but we cannot prove that they are actually invariants. On the other hand, in the static approach, invariants hold for any possible execution of the program in any possible environment, but feasible solutions are often very approximate. The more precise a static analysis is, the more the computational complexity grows, and very precise analyses are infeasible in practice.

We believe that combining the two approaches, we can overcome the drawbacks of both. In the literature, it is very common to find specific static analysis helping dynamic analysis (for instance, in

E-mail addresses: amato@sci.unich.it (G. Amato), parton@sci.unich.it (M. Parton), scozzari@sci.unich.it (F. Scozzari).

0747-7171/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2011.12.052

the field of runtime verification), but we propose the opposite, that is to help static analysis by means of dynamic information. We first observe the dynamic behavior of the program and analyze it with statistical methods, and then we use the collected information to drive a subsequent static analysis based on the abstract interpretation framework (Cousot and Cousot, 1979, 1992). In particular, we focus on invariants in the form of linear inequalities.

1.1. The static side

The idea of abstract interpretation is to replace the (concrete) semantics of a program with an abstract semantics, computed over a domain of abstract objects. The concrete semantics is specified by means of a *concrete domain* C and some basic operators. Typical operators for imperative programs include assignment, test and union operators, which are used in the semantics for treating, respectively, the basic statements for variable assignment, *if–then–else* and loops.

An abstract interpretation is specified by a set A of *abstract objects*. Abstract objects describe the properties of the system we are interested in. The relationship between concrete and abstract objects is formalized by an abstraction map $\alpha : C \to A$ which, given a concrete state $c \in C$, yields the most precise property $a \in A$ which holds in the state c. An *abstract domain* is given by the set A of the abstract objects and the abstraction map α.

For instance, consider the concrete domain $\wp(\mathbb{Z})$ and the abstract domain $\mathsf{Int}_{\mathbb{Z}} = \{[a, b] | a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}\}$ of (possibly unbounded) intervals. The intuition is that an abstract object $[3, 8]$ represents the set of integer numbers $\{3, 4, 5, 6, 7, 8\}$. This may be formalized by defining the abstraction map $\alpha(X) = [\inf X, \sup X]$, so that, for example, we have $\alpha(\{3, 4, 5, 8\}) = [3, 8]$.

The goal of any abstract interpretation is to compute an abstract semantics formally derived from the concrete semantics, by replacing each basic operator with an abstract, sound operator, working with abstract objects. Consider the operator $inc : \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ such that $inc(X) = \{n + 1 \mid n \in X\}$, which intuitively corresponds to the program statement `x = x+1`. The best correct approximation of inc is $inc^{\alpha}([a, b]) = [a + 1, b + 1]$ where we assume that $-\infty + 1 = -\infty$ and $+\infty + 1 = +\infty$.

The theory of abstract interpretation ensures us that the abstract semantics correctly approximates the concrete semantics, so that properties proved on the abstract side hold also on the concrete side. Moreover, given an abstract domain A and a concrete operator $f : C \to C$, it is always possible to design a best correct abstract operator corresponding to f, which is uniquely identified by the abstract domain A. Thus, the abstract domain is the key notion in the choice of any abstract interpretation.

1.2. Numerical abstract domains

In the literature, we find many abstract domains for numerical properties. The simplest one is the interval domain (Cousot and Cousot, 1976). Here, each set of real (rational, integer) numbers is approximated by a (possibly unbounded) interval $[a, b]$. The expressive power of the domain is very limited, since we can only find invariants of the form $a \leq x \leq b$ where x is a program variable. For this reason, analysis with the interval domain are often very imprecise, but enjoy a low computational complexity.

A very rich domain is that of convex polyhedra (Cousot and Halbwachs, 1978), where we can express any linear inequality such as $a_1x_1 + \cdots + a_nx_n \leq b$. Unfortunately, the computational complexity of the domain is, in most cases, prohibitive.

Many other numerical abstract domains strive to trade the accuracy of convex polyhedra for higher speed, by considering only convex polyhedra of fixed shapes, such as octagons (Miné, 2006), weighted hexagons (Fulara et al., 2010) and two variables per inequality (Simon et al., 2003). This reduces both the complexity and the expressive power of the domains.

The precision of the analyses may often be improved with the use of special-purpose abstract domains, such as the domains for the analysis of digital filters (Feret, 2004), or the arithmetic–geometric progression abstract domain (Feret, 2005). This idea may be pushed further by devising domains not just for a class of applications, but for a single program, following the intuition that, if we know the general form of the `while`-loop invariants which occur in a program, domains able to express these invariants should reach a higher precision than others. This idea is developed in the

```
xyline = function(x)
{
   assume(x>=0)
   y = -x
   while(x>y) {
 ①  x = x-1
    y = y+1
   }
}
```

Fig. 1. The example program `xyline`.

x	y
10	-10
9	-9
8	-8
7	-7
6	-6
5	-5

Fig. 2. A partial execution trace of the example program.

template polyhedra approach (Sankaranarayanan et al., 2005), where the analysis is performed by using a (finite) number of convex polyhedra (e.g., one for each program point) whose shape is fixed a priori. This amounts to saying that we fix *a priori* a finite set of linear forms $\bar{a}_1x_1 + \cdots + \bar{a}_nx_n$ (the template) for each program point, and the analyzer finds out the correct bound b such that $\bar{a}_1x_1 + \cdots + \bar{a}_nx_n \leq b$. The idea is appealing, but it lacks an effective method to choose the template and, due to the general form of the polyhedra, linear programming is needed to compute the abstract operators.

1.3. The dynamic side

We strongly believe that many drawbacks of the numerical abstract domains could be offset by first performing a dynamic analysis of the program behavior. Consider the program in Fig. 1 where the parameter x is the input and y is a local variable, and its partial execution trace for the input $x = 10$. Collecting the values for the variables x and y at program point ① for the first 6 iterations of the `while`-statement, we obtain Fig. 2. If we abstract this set of values in the interval domain, we get the shaded area in Fig. 3, given by

$$\begin{cases} 5 \leq x \leq 10, \\ -10 \leq y \leq -5. \end{cases}$$

The key point is that the abstraction in the interval domain depends on the coordinate system we choose to draw the boxes. With the standard choice of (x, y) as a coordinate system, the box in Fig. 3 is a very rough approximation of the partial trace, but we can improve the precision by conveniently changing the axes. For instance, consider a different coordinate system whose axes (x', y') are clockwise rotated by 30°. The abstraction in this "rotated interval domain" is depicted in Fig. 4. The two boxes in Fig. 4 are incomparable as sets of points, nonetheless the rotated box seems to fit better: for example, it has a smaller area. The issue is how to find a "best rotation".

1.4. The statistical methods

We apply to the sample data a statistical technique called *orthogonal simple component analysis* (OSCA), recently proposed by Anaya-Izquierdo et al. (2001), which is a variant of *principal component*

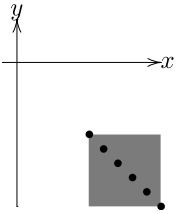

Fig. 3. Interval abstraction of a partial execution trace, observed at program point ①.

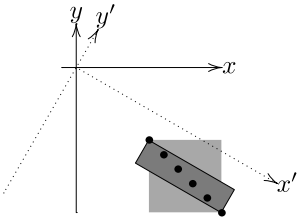

Fig. 4. Abstraction with boxes rotated by 30°.

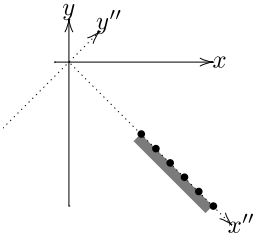

Fig. 5. Abstraction with boxes rotated by 45°.

analysis (PCA). Given an observed sample of data (points in $\mathbb{R}^n$), the intuitive idea of PCA is to find a new orthonormal coordinate system maximizing the variance of the collected values. More explicitly, PCA finds new pairwise orthogonal axes, called *principal components*, such that the variance of the projection of the data on the first axis is the maximum among all possible directions, the variance of the projection of the data on the second axis is the maximum among all possible directions which are orthogonal to the first axis, and so on. In our example, the greatest variance is obtained by projecting the data (the values in Fig. 2) along the line $y = -x$. Therefore, the first component found by PCA is the vector $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ (which corresponds to the line $y = -x$) and the second one is $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$.

The vectors computed by the PCA cannot be directly used in the static analysis, due to approximation errors which may cause a great loss of accuracy (see Example 10 at page 22). Thus we apply OSCA, which returns pairwise orthogonal vectors approximating PCA, but having small integer coefficients. This property also helps the correct implementation of the rotated interval domains. For the program in Fig. 1, OSCA finds the vectors $(1, -1)$ and $(1, 1)$ which correspond to the axes (x'', y'') in Fig. 5, that is a clockwise rotation of 45°. The values in the partial trace are approximated by the box

$$\begin{cases} 5 \leq x'' \leq 10, \\ 0 \leq y'' \leq 0, \end{cases}$$

which may be interpreted in the original coordinates (x, y) as

$$\begin{cases} 10 \leq x - y \leq 20, \\ 0 \leq x + y \leq 0. \end{cases}$$

It is worth noting that, in our example, the `while` invariant at the program point ① is $x + y = 0, x - y \geq 0$, and it may be expressed with boxes only when they are rotated by 45°. Thus, an abstract interpretation-based analyzer using rotated boxes as abstract objects, could infer this invariant.

More generally, this suggests that, if we consider well-known numerical abstract domains and adapt them to work with non-standard coordinate systems, we can improve the precision of the analysis without much degradation of performance. In this paper we develop the theoretical foundation and the implementation to validate this intuition, using the interval domain as a case study. In Section 8, we will show that our analysis actually infers the invariant $x + y = 0, x - y \geq 0$.

The paper is structured as follows. Section 2 introduces some notation. Section 3 defines the concrete domain and its operators, while in Section 4 we recall the abstract domain of intervals. The new rotated interval domains are presented in Section 5. Section 6 introduces OSCA, used to automatically infer the best rotated interval domain for the given program, and we discuss in Section 7 a number of possible optimizations and extensions. Section 8 presents the implementation and shows some experimental results. In Section 9 we discuss related work in the literature and Section 10 concludes the paper presenting possible directions for future work. This paper is a revised and extended version of (Amato et al., 2010a).

2. Notation

2.1. Linear algebra

We denote by $\bar{\mathbb{R}}$ the ordered field of real numbers extended with $-\infty$ and $+\infty$, and we use boldface for elements of $\mathbb{R}^n$ and $\bar{\mathbb{R}}^n$. Given $\boldsymbol{u}, \boldsymbol{v} \in \bar{\mathbb{R}}^n$ and a relation $\bowtie \in \{<, >, \leq, \geq\}$, we write $\boldsymbol{u} \bowtie \boldsymbol{v}$ if and only if $u_i \bowtie v_i$ for each $i \in \{1, \ldots, n\}$. We denote the scalar product on $\mathbb{R}^n$ by $\boldsymbol{u} \cdot \boldsymbol{v} \stackrel{\text{def}}{=} u_1 v_1 + \cdots + u_n v_n$, and the length of $\boldsymbol{v} \in \mathbb{R}^n$ by $|\boldsymbol{v}| \stackrel{\text{def}}{=} \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$.

If $A = (a_{ij})$ is any matrix, we denote by A^T its *transpose*. If A is invertible, A^{-1} denotes its inverse, and $\mathrm{GL}(n)$ is the group of $n \times n$ invertible matrices. The identity matrix in $\mathrm{GL}(n)$ is denoted by I_n, and any $A \in \mathrm{GL}(n)$ such that $AA^T = I_n$ is called an *orthogonal* matrix. Looking at vectors as $n \times 1$ matrices, and vice versa, we denote by $(\boldsymbol{v}_1 | \ldots | \boldsymbol{v}_n)$ the matrix whose columns are the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. Moreover, we denote by A_{j*} the $1 \times n$ matrix given by the j-th row of A, and by A_{*j} the $n \times 1$ matrix given by the j-th column of A. We have $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v}$, whereas $\boldsymbol{u}\boldsymbol{v}^T$ is a $n \times n$ matrix.

A matrix $A = (\boldsymbol{v}_1 | \ldots | \boldsymbol{v}_n)$ is orthogonal if and only if $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are *orthonormal*, namely, they have length 1 and are pairwise orthogonal. The standard orthonormal basis of $\mathbb{R}^n$ is denoted by $\{\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n\}$, and any matrix $A \in \mathrm{GL}(n)$ can be viewed as a change of basis, mapping $\boldsymbol{e}^i$ to the i-th column of A, that is, $(\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n) \mapsto (A\boldsymbol{e}^1, \ldots, A\boldsymbol{e}^n)$. If $(x_1, \ldots, x_n) \in \mathbb{R}^n$ are coordinates of a vector with respect to $\{A\boldsymbol{e}^1, \ldots, A\boldsymbol{e}^n\}$, then $A(x_1, \ldots, x_n)^T$ are the coordinates of the same vector with respect to $\{\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n\}$.

2.2. Abstract interpretation

Given complete lattices $(C, \leq_C)$ and $(A, \leq_A)$, respectively called the *concrete domain* and the *abstract domain*, a *Galois connection* is a pair (α, γ) of monotone maps $\alpha : C \to A, \gamma : A \to C$ such that $\alpha\gamma \leq_A \mathrm{id}_A$ and $\gamma\alpha \geq_C \mathrm{id}_C$. If $\alpha\gamma = \mathrm{id}_A$, then (α, γ) is called a *Galois insertion*. Given a monotone map $f : C \to C$, the map $\tilde{f} : A \to A$ is a *correct approximation* of f if $\alpha f \leq \tilde{f}\alpha$. The *best correct approximation* of f is the smallest correct approximation f^α of f. It is well-known that $f^\alpha = \alpha f \gamma$. When $f^\alpha \alpha = \alpha f$ then f is called α-*complete*. See Cousot and Cousot (1992) for further details.

3. Analysis of numerical properties: the concrete domain

In this paper we are faced with the problem of discovering, for each program point, a property of the value of numerical variables which is guaranteed to hold for all the executions of the program and for any possible input. Since any numerical property involving n variables may be represented as a subset of $\mathbb{R}^n$, a very precise analysis may be obtained, at least conceptually, by manipulating

these subsets through a limited collection of *basic operators*, corresponding to syntactic constructs such as assignments, conditionals, loops, etc.... Every other analysis may be obtained as an abstract interpretation of $\wp(\mathbb{R}^n)$.

We start by presenting, in this section, the basic concrete operators needed to design static analyses of imperative programs. In the next two sections, we will define the corresponding abstract operators for two abstractions of $\wp(\mathbb{R}^n)$. For details on how to build an abstract interpreter from the basic operators, the reader may refer to Cousot and Cousot (1976) and Cousot (1999).

3.1. Lattice-theoretic operators

The set of *concrete properties* $\wp(\mathbb{R}^n)$ is a complete lattice with the standard set-theoretic operations of *union* and *intersection*, and with ordering given by the subset relationship. The *least element* is $\emptyset$ and the *greatest element* is $\mathbb{R}^n$. In the following, $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ denotes the vector of program variables, for a fixed $n \in \mathbb{N}$, and $\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$ denote parameters defining linear constraints.

3.2. Linear assignment

The linear assignment operator is used to analyze the behavior of the statement $\mathtt{x_i} = \mathtt{expr} + \mathtt{b}$ where `expr` is a linear expression on the variables of the program and `b` is a constant. The *linear assignment* operator $\mathsf{assign}(i, \boldsymbol{a}, b) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ is the pointwise extension of:

$$\mathsf{assign}(i, \boldsymbol{a}, b)(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{y} \quad \text{where } y_j = \begin{cases} x_j & \text{if } j \neq i\,, \\ \boldsymbol{a} \cdot \boldsymbol{x} + b & \text{if } j = i\,. \end{cases} \tag{1}$$

3.3. Forget

The *forget* operator $\mathsf{forget}(i) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ models non-deterministic assignment to the variable x_i. It loses all the information regarding the i-th variable, while keeping all the information for the remaining variables. It is formally defined as:

$$\mathsf{forget}(i)(X) \stackrel{\text{def}}{=} \{(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \mid \boldsymbol{x} \in X, y \in \mathbb{R}\}. \tag{2}$$

The forget operator is quite useful as a fall-back assignment in all those cases where we cannot (or we do not want to) analyze more precisely, such as for non-linear assignments or calls to unknown functions.

3.4. Test

The *test* operator $\mathsf{test}(\boldsymbol{a}, b, \bowtie) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ corresponds to the *then*-branch of the `if`-statement "`if` $(\boldsymbol{a} \cdot \boldsymbol{x} \bowtie b)$", where $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$. It is defined as

$$\mathsf{test}(\boldsymbol{a}, b, \bowtie)(X) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in X \mid \boldsymbol{a} \cdot \boldsymbol{x} \bowtie b\}.$$

Every conditional statement, even if it contains Boolean operations, may be analyzed using only test together with lattice-theoretic operators.

3.5. Other basic operators

We briefly discuss other operators which may be useful for the static analysis of numerical properties.

Backward assignment. It is used for backward abstract interpretation and, in some settings, for refining the analysis of conditional statements (Miné, 2004). We do not consider backward abstract interpretation in this paper, and the abstract operators for $\mathsf{test}(\boldsymbol{a}, b, \bowtie)$ will be precise enough not to require backward assignment.

Non-linear assignments and tests. These are generally handled with a variety of techniques such as linearization (Miné, 2006), approximation with interval arithmetic or with the forget operator (Miné, 2006; Cousot and Halbwachs, 1978), and special-purpose assignment operators for particular kinds of non-linear assignments (Feret, 2004, 2005). It is a problem which is orthogonal to the development of the abstract domain, and any of these techniques can be applied to our domains as well.

4. The interval domain

In this section we recall the basic definitions and operators of the *interval domain* (Cousot and Cousot, 1976) as an abstraction of $\wp(\mathbb{R}^n)$. The analysis using this domain is generally called *range analysis*, and abstract elements are called *boxes*.

4.1. Abstract domain

A set $\mathcal{B} \subseteq \mathbb{R}^n$ is called a (closed) *box* if there are *bounds* $\boldsymbol{m}, \boldsymbol{M} \in \bar{\mathbb{R}}^n$ such that

$$\mathcal{B} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{m} \leq \boldsymbol{x} \leq \boldsymbol{M}\}. \tag{3}$$

Each box $\mathcal{B}$ determines a pair $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$ of vectors in $\bar{\mathbb{R}}^n$. Conversely, each pair $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$ determines a box $\mathcal{B}$ according to (3). This correspondence is not one-to-one since different choices of $\boldsymbol{m}$ and $\boldsymbol{M}$ represent the empty box. We fix $\bot \stackrel{\text{def}}{=} \langle +\infty, -\infty \rangle$ as the unique representation for the empty box, where $+\infty$ (resp. $-\infty$) is the vector whose elements are all $+\infty$ (resp. $-\infty$). This allows us to define the *interval domain* as

$$\mathsf{Int} \stackrel{\text{def}}{=} \{\langle \boldsymbol{m}, \boldsymbol{M} \rangle \in \bar{\mathbb{R}}^n \times \bar{\mathbb{R}}^n \mid \boldsymbol{m} \leq \boldsymbol{M}, \forall i.\, m_i \neq +\infty, M_i \neq -\infty\} \cup \{\bot\}. \tag{4}$$

With an abuse of terminology, we use the term box for elements of Int. The domain Int may be ordered with respect to set inclusion:

$$\langle \boldsymbol{m}, \boldsymbol{M} \rangle \leq \langle \boldsymbol{m}', \boldsymbol{M}' \rangle \Leftrightarrow \boldsymbol{m} \geq \boldsymbol{m}' \wedge \boldsymbol{M} \leq \boldsymbol{M}'. \tag{5}$$

This gives rise to a Galois insertion $(\alpha^{\mathsf{Int}}, \gamma^{\mathsf{Int}}) : \wp(\mathbb{R}^n) \rightleftharpoons \mathsf{Int}$ defined as follows:

$$\gamma^{\mathsf{Int}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{m} \leq \boldsymbol{x} \leq \boldsymbol{M}\}, \tag{6}$$

$$\alpha^{\mathsf{Int}}(X) \stackrel{\text{def}}{=} \langle \boldsymbol{m}', \boldsymbol{M}' \rangle, \tag{7}$$

where, for each $i \in \{1, \ldots, n\}$,

$$m_i' \stackrel{\text{def}}{=} \inf_{\boldsymbol{x} \in X} x_i, \qquad M_i' \stackrel{\text{def}}{=} \sup_{\boldsymbol{x} \in X} x_i.$$

Range analysis has its roots in earlier work on interval arithmetic, originated in the field of numerical analysis. Therefore, we briefly recall the relevant notions on interval arithmetic which will be used throughout the paper.

4.2. Interval arithmetic

Rounding errors and measurement errors in mathematical computations can be bounded by using *interval arithmetic*. The idea is to replace an unknown real number x with an interval $[a, b]$ such that $x \in [a, b]$. The key point is the ability to use floating point numbers as bounds of intervals. The reader may consult (Hickey et al., 2001) for a rigorous presentation of interval arithmetic and (Kearfott, 1996) for a survey on its applications.

Given $a, b \in \bar{\mathbb{R}}$, we denote by $[a, b]$ the interval $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$. If $a > b$ then $[a, b] = \emptyset$. We use $\mathbb{I}$ for the set of intervals and the variable δ to vary over $\mathbb{I}$. We denote with $\underline{\delta}$ the lower bound of δ and with $\overline{\delta}$ its upper bound. Therefore, $\delta = [\underline{\delta}, \overline{\delta}]$.

It turns out that most arithmetic operations, when extended pointwise to sets, map intervals to intervals. This holds, in particular, for addition, subtraction and multiplication. Moreover, the bounds

of the results may be easily computed from the bounds of the arguments. For non-empty bounded intervals we have that:

$$[a, b] + [c, d] = [a + c, b + d], \tag{8}$$

$$[a, b] - [c, d] = [a - d, b - c], \tag{9}$$

$$[a, b] * [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]. \tag{10}$$

All interval operations are strict, i.e., the result is $\emptyset$ as soon as one of its arguments is $\emptyset$. The cases for unbounded intervals may be found in Hickey et al. (2001).

Interval vectors and *interval matrices* are the counterparts of real vectors and matrices, when elements are intervals instead of real numbers. Formally, an interval vector is an element of $\mathbb{I}^n$ for some n, while an interval matrix is an element of $\mathbb{I}^{n\times m}$. We use $\boldsymbol{\delta}$ to denote interval vectors and Δ to denote interval matrices. Given two vectors $\boldsymbol{m}, \boldsymbol{M} \in \bar{\mathbb{R}}^n$, we denote by $[\boldsymbol{m}, \boldsymbol{M}]$ the interval vector $\boldsymbol{\delta}$ such that $\delta_i = [m_i, M_i]$. We denote by $\underline{\boldsymbol{\delta}}$ (resp. $\overline{\boldsymbol{\delta}}$) the vector of the lower bounds (resp. upper bounds) of $\boldsymbol{\delta}$, so that $\boldsymbol{\delta} = [\underline{\boldsymbol{\delta}}, \overline{\boldsymbol{\delta}}]$. Finally, we use $\boldsymbol{x} \in \boldsymbol{\delta}$ as a short form of $\underline{\boldsymbol{\delta}} \leq \boldsymbol{x} \leq \overline{\boldsymbol{\delta}}$. The notation extends naturally to interval matrices.

Standard operations for vectors and matrices such as sum, row-by-column product, scalar product, may be performed on interval vectors and matrices just replacing standard arithmetic operations with their interval counterparts. In the context of an interval expression, a real number x, a vector $\boldsymbol{x}$ or a matrix A should be considered as a short form for $[x, x]$, $[\boldsymbol{x}, \boldsymbol{x}]$ or $[A, A]$ respectively. Operations on interval vectors and matrices are *safe*. For example, if $\boldsymbol{\delta}$ is an interval vector, Δ an interval matrix, $\boldsymbol{x} \in \boldsymbol{\delta}$ and $A \in \Delta$, then $A\boldsymbol{x} \in \Delta\boldsymbol{\delta}$. Note, however, that standard properties of linear algebra do not necessarily hold.

Observe that boxes and interval vectors are two different representations of the same mathematical object: the box $\langle \boldsymbol{m}, \boldsymbol{M}\rangle$ corresponds to the interval vector $[\boldsymbol{m}, \boldsymbol{M}]$. Again, the correspondence may be made one-to-one with appropriate restrictions to $\boldsymbol{m}$ and $\boldsymbol{M}$. Therefore, in the following we view boxes either as pairs of vectors of extended reals, or as interval vectors.

4.3. Lattice-theoretic operators

The best correct approximation of set union $\boldsymbol{\delta} \cup^{\mathsf{Int}} \boldsymbol{\delta}'$ yields the smallest box containing both $\boldsymbol{\delta}$ and $\boldsymbol{\delta}'$. It is given by

$$\boldsymbol{\delta} \cup^{\mathsf{Int}} \boldsymbol{\delta}' \stackrel{\text{def}}{=} \boldsymbol{\delta}'',$$

where

$$\underline{\delta}_i'' \stackrel{\text{def}}{=} \min(\underline{\delta}_i, \underline{\delta}_i') \qquad \overline{\delta}_i'' \stackrel{\text{def}}{=} \max(\overline{\delta}_i, \overline{\delta}_i'). \tag{11}$$

The abstract intersection operator $\boldsymbol{\delta} \cap^{\mathsf{Int}} \boldsymbol{\delta}'$ may be computed as for $\cup^{\mathsf{Int}}$, swapping the minimum and maximum operations. Abstract union and intersection are actually the lowest upper bound and greatest lower bound of Int. The computation complexity of both operators is $O(n)$.

4.4. Linear assignment, forget and test

Since all abstract operators are strict, we will provide explicit definitions only for a non-empty input argument $\boldsymbol{\delta} \neq \bot$.

Linear assignment. The best correct approximation of $\mathsf{assign}(i, \boldsymbol{a}, b)$ is given by

$$\mathsf{assign}^{\mathsf{Int}}(i, \boldsymbol{a}, b)(\boldsymbol{\delta}) \stackrel{\text{def}}{=} \boldsymbol{\delta}',$$

where

$$\delta_j' \stackrel{\text{def}}{=} \begin{cases} \delta_j & \text{if } i \neq j, \\ \boldsymbol{a} \cdot \boldsymbol{\delta} + b & \text{if } i = j. \end{cases} \tag{12}$$

Note that $\boldsymbol{a} \cdot \boldsymbol{\delta} + b$ is computed according to interval arithmetic. The computational complexity is $O(n)$.

Forget. The best correct approximation of $\mathsf{forget}(i)$ is given by

$$\mathsf{forget}^{\mathsf{Int}}(i)(\boldsymbol{\delta}) \stackrel{\text{def}}{=} \boldsymbol{\delta}',$$

where

$$\delta'_j \stackrel{\text{def}}{=} \begin{cases} \delta_j & \text{if } i \neq j, \\ [-\infty, +\infty] & \text{otherwise.} \end{cases} \tag{13}$$

The computational complexity is $O(n)$ if implemented by creating a new box object, $O(1)$ if in-place replacement is used.

Test. We first consider the case when $\bowtie$ is $\leq$. The best correct approximation of $\mathsf{test}(\boldsymbol{a}, b, \leq)$ is given by

$$\mathsf{test}^{\mathsf{Int}}(\boldsymbol{a}, b, \leq)(\boldsymbol{\delta}) \stackrel{\text{def}}{=} \boldsymbol{\delta}',$$

where

$$\delta'_i \stackrel{\text{def}}{=} \begin{cases} [\underline{\delta}_i, \min(c_i, \overline{\delta}_i)] & \text{if } a_i > 0, \\ [\max(C_i, \underline{\delta}_i), \overline{\delta}_i] & \text{if } a_i < 0, \\ \delta_i & \text{if } a_i = 0, \end{cases} \tag{14}$$

and

$$[c_i, C_i] \stackrel{\text{def}}{=} \left(b - \sum_{j \neq i} a_j \delta_j \right) / a_i.$$

This procedure has complexity $O(n)$, since we may compute $[c, C] \stackrel{\text{def}}{=} b - \boldsymbol{a} \cdot \boldsymbol{\delta}$ only once, and obtain $[c_i, C_i]$ by removing the effect of $a_i \delta_i$ in $[c, C]$.

The case for $\geq$ is symmetric, while the result for $=$ may be obtained as the intersection of the results for $\leq$ and $\geq$. The result of $\mathsf{test}^{\mathsf{Int}}(\boldsymbol{a}, b, <)(\boldsymbol{\delta})$ is either $\bot$ if $\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \subseteq \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \geq b\}$, or the input box $\boldsymbol{\delta}$ otherwise. To test whether $\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \subseteq \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \geq b\}$, it is enough to check by interval arithmetic that $\boldsymbol{a} \cdot \boldsymbol{\delta} \leq [b, +\infty]$. The case for $>$ is symmetric. Finally, the result for $\neq$ may be obtained as the abstract union of the results for $<$ and $>$.

4.5. Ensuring termination of the analysis

Since the interval domain has infinite ascending chains, and the analysis requires a fixpoint computation, the standard tools of widening/narrowing are used to ensure the termination of the analysis (Cousot and Cousot, 1976, 1992). The intuitive idea is to compare the result of the previous iteration to the current one. Whenever the current iteration enlarges a bound, then the widening immediately sets it to $+\infty$ (resp. $-\infty$). For instance, if the result of the previous iteration is the interval $[3, 4]$ and the result of the current iteration is $[3, 5]$, then the widening yields $[3, +\infty]$. When a fixpoint is reached, this is generally not the least fixpoint. A descending iteration with narrowing may be used to improve the result. During a descending iteration, the narrowing only improves infinite bounds. For instance, if the result of the previous iteration is the interval $[-1, +\infty]$ and the result of the current iteration is $[0, 10]$, then the narrowing yields $[-1, 10]$.

5. The parallelotope domains

Analyses using the abstract domain of intervals are generally very inaccurate, due to the limited expressive power of the abstract objects, but in contrast they are very fast. Our proposal is to use boxes as abstract objects, but interpreted in a different coordinate system, in order to fit the original data with a higher precision than with standard boxes. Every choice of a matrix $A \in \mathrm{GL}(n)$ gives a new coordinate system in $\mathbb{R}^n$, and boxes with respect to this transformed coordinates are called *parallelotopes*. Remark that we are not restricting to orthogonal changes of basis: this means that we consider any invertible linear transformation, such as rotation, reflection, stretching, compression, shear or any combination of these.

Example 1. Consider the set $X = \{(u, -u) \mid u \geq 0\} \subseteq \mathbb{R}^2$ corresponding to the invariant $x + y = 0$, $x - y \geq 0$ of the program in Fig. 1. If we directly abstract X in the interval domain, we get $\alpha^{\mathsf{Int}}(X) = \langle (0, -\infty), (+\infty, 0)\rangle = \mathbb{R}^+ \times \mathbb{R}^-$, with a sensible loss of precision. Let us consider a clockwise rotation of 45°, centered on the origin, of the standard coordinate system. The matrix

$$A = \begin{bmatrix} \cos\left(-\frac{\pi}{4}\right) & \sin\left(-\frac{\pi}{4}\right) \\ -\sin\left(-\frac{\pi}{4}\right) & \cos\left(-\frac{\pi}{4}\right) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

transforms the standard coordinates into the rotated coordinates.

We want to abstract X with boxes on the rotated coordinate system. To this aim, we first compute the rotated coordinates of the points in X, and then compute the smallest enclosing box. Since the rotated coordinates are given by $A(x, y)^T$, we obtain:

$$\begin{aligned} \alpha^{\mathsf{Int}}(AX) &= \alpha^{\mathsf{Int}}(\{A\boldsymbol{v} \mid \boldsymbol{v} \in X\}) \\ &= \alpha^{\mathsf{Int}}\left(\left\{\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}\begin{bmatrix} u \\ -u \end{bmatrix} \mid u \in \mathbb{R}^+\right\}\right) \\ &= \alpha^{\mathsf{Int}}\left(\left\{\begin{bmatrix} u\sqrt{2} \\ 0 \end{bmatrix} \mid u \in \mathbb{R}^+\right\}\right) = \langle (0, 0), (+\infty, 0)\rangle. \end{aligned}$$

The box $\langle (0, 0), (+\infty, 0)\rangle$ computed above may be represented algebraically in the standard coordinate system as

$$(0, 0) \leq A(x, y)^T \leq (+\infty, 0)$$

or, more explicitly, as

$$\begin{cases} 0 \leq \frac{x}{\sqrt{2}} - \frac{y}{\sqrt{2}} \leq +\infty, \\ 0 \leq \frac{x}{\sqrt{2}} + \frac{y}{\sqrt{2}} \leq 0. \end{cases}$$

More in general, using the matrix A, we may represent all the parallelotopes of the form

$$\begin{cases} m_1 \leq \frac{x}{\sqrt{2}} - \frac{y}{\sqrt{2}} \leq M_1, \\ m_2 \leq \frac{x}{\sqrt{2}} + \frac{y}{\sqrt{2}} \leq M_2, \end{cases}$$

or, equivalently,

$$\begin{cases} m_1 \leq x - y \leq M_1, \\ m_2 \leq x + y \leq M_2. \end{cases}$$

Thus, we have transformed a non-relational analysis into a relational one, where the form of the relationships is given by the matrix A. If we concretize the box by applying γ^{Int} and using the matrix A to convert the result to the standard coordinate system, we obtain $A^{-1}\gamma^{\mathsf{Int}}\alpha^{\mathsf{Int}}(AX) = X$. Thus, we get much better precision than using standard boxes. We stress out that we need to choose A cleverly, on the base of the specific data set, otherwise we may lose precision: for example, if $X' = \{(u, 0) \mid u \in \mathbb{R}\}$, then $\gamma^{\mathsf{Int}}(\alpha^{\mathsf{Int}}(X')) = X'$ but $A^{-1}\gamma^{\mathsf{Int}}\alpha^{\mathsf{Int}}(AX') = \mathbb{R}^2$.

It is worth noting that, if we prefer not to deal with irrational numbers, we may choose the transformation matrix

$$A' = \sqrt{2}\,A = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}.$$

This corresponds to a 45-degree clockwise rotation followed by a $\sqrt{2}$ scaling in all directions. □

5.1. Abstract domains of parallelotopes

We define the abstract domains of parallelotopes by using the same complete lattice Int of the interval domain, but equipped with a different abstraction function, and different abstract operators.

Definition 2 (*The Parallelotope Domains*)**.** Given $A \in \mathrm{GL}(n)$, we define the maps $\gamma^A : \mathsf{Int} \to \wp(\mathbb{R}^n)$ and $\alpha^A : \wp(\mathbb{R}^n) \to \mathsf{Int}$ as

$$\gamma^A(\boldsymbol{\delta}) \stackrel{\text{def}}{=} A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}),$$
$$\alpha^A(X) \stackrel{\text{def}}{=} \alpha^{\mathsf{Int}}(AX).$$

It is easy to check that it yields a Galois insertion $(\alpha^A, \gamma^A) : \wp(\mathbb{R}^n) \rightleftharpoons \mathsf{Int}$. Intuitively, the abstraction α^A first projects the points into the new coordinate system, then computes the standard box abstraction. The concretization map γ^A performs the opposite process. Remark that, as a particular case, we have $\alpha^{I_n} = \alpha^{\mathsf{Int}}$ and $\gamma^{I_n} = \gamma^{\mathsf{Int}}$.

In the rest of this section we present the abstract operators for the parallelotope domains. In most cases, they can be easily recovered from the corresponding operators on the interval domain.

In the computational complexity of all the abstract operators, we always ignore the cost for inverting A. If A is orthogonal, it may be considered constant since $A^{-1} = A^T$ and we can perform transposition "on the fly" when needed. If A is not orthogonal, the inverse can be computed with standard algorithms whose complexity is between quadratic and cubic. However, A^{-1} needs to be computed only once for the entire execution of the abstract interpretation procedure, hence its computational cost is much less relevant then the cost of the abstract operators.

The issues regarding the correct implementation of these operators with machine arithmetic will be considered in Section 5.6.

5.2. Lattice-theoretic operators

We denote by $\cup^A$ and $\cap^A$ the best correct approximation of concrete union and intersection respectively.

Theorem 3 (*Union and Intersection*)**.** *Given $\boldsymbol{\delta}, \boldsymbol{\delta}' \in \mathsf{Int}$, we have that:*

$$\boldsymbol{\delta} \cup^A \boldsymbol{\delta}' = \boldsymbol{\delta} \cup^{\mathsf{Int}} \boldsymbol{\delta}', \qquad \boldsymbol{\delta} \cap^A \boldsymbol{\delta}' = \boldsymbol{\delta} \cap^{\mathsf{Int}} \boldsymbol{\delta}'.$$

The computational complexity of both operators is $O(n)$.

Proof. We recall that

$$\boldsymbol{\delta} \cup^A \boldsymbol{\delta}' \stackrel{\text{def}}{=} \alpha^A(\gamma^A(\boldsymbol{\delta}) \cup \gamma^A(\boldsymbol{\delta}')).$$

By replacing α^A and γ^A with their definitions, A and A^{-1} cancel out and we have that $\cup^A$ is the same as $\cup^{\mathsf{Int}}$. The same holds for intersection. □

Since parallelotopes use the same abstract domain as boxes with the same ordering, we may also reuse the same widening/narrowing operators.

5.3. Linear assignment

We denote by $\mathsf{assign}^A(i, \boldsymbol{a}, b)$ the best correct approximation of the concrete linear assignment $\mathsf{assign}(i, \boldsymbol{a}, b)$. Since the concrete assignment is strict, we have that $\mathsf{assign}^A(i, \boldsymbol{a}, b)(\bot) = \bot$. The same holds for the forget and test operators. In the following we will provide explicit definition for the non-empty cases only.

Theorem 4 (*Assignment*). *Given* $\bot \neq \boldsymbol{\delta} \in \mathsf{Int}$, *we have that*

$$\mathsf{assign}^A(i, \boldsymbol{a}, b)(\boldsymbol{\delta}) = H_{i,\boldsymbol{a}}\boldsymbol{\delta} + Ab\boldsymbol{e}^i,$$

where

$$H_{i,\boldsymbol{a}} \stackrel{\text{def}}{=} I_n + A\boldsymbol{e}^i(\boldsymbol{a}^T - \boldsymbol{e}^{iT})A^{-1}.$$

The computational complexity is $O(n^2)$.

Proof. Note that $\mathsf{assign}(i, \boldsymbol{a}, b)$ may be rewritten as

$$\mathsf{assign}(i, \boldsymbol{a}, b)(\boldsymbol{x}) = Z_{i,\boldsymbol{a}}\boldsymbol{x} + b\boldsymbol{e}^i,$$

where

$$\begin{aligned} Z_{i,\boldsymbol{a}} &\stackrel{\text{def}}{=} I_n - \boldsymbol{e}^i\boldsymbol{e}^{iT} + \boldsymbol{e}^i\boldsymbol{a}^T \\ &= I_n + \boldsymbol{e}^i(\boldsymbol{a}^T - \boldsymbol{e}^{iT}). \end{aligned}$$

Accordingly, we may rewrite the abstract operator $\mathsf{assign}^A(i, \boldsymbol{a}, b)$ as:

$$\begin{aligned} \alpha^A(\mathsf{assign}(i, \boldsymbol{a}, b)(\gamma^A(\boldsymbol{\delta}))) &= \alpha^{\mathsf{Int}}(A\ \mathsf{assign}(i, \boldsymbol{a}, b)(A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}))) \\ &= \alpha^{\mathsf{Int}}(AZ_{i,\boldsymbol{a}}A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) + Ab\boldsymbol{e}^i) \\ &= \alpha^{\mathsf{Int}}(H_{i,\boldsymbol{a}}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) + Ab\boldsymbol{e}^i), \end{aligned} \tag{15}$$

where $H_{i,\boldsymbol{a}} \stackrel{\text{def}}{=} AZ_{i,\boldsymbol{a}}A^{-1} = I_n + A\boldsymbol{e}^i(\boldsymbol{a}^T - \boldsymbol{e}^{iT})A^{-1}$.

We need to remove α^{Int} and γ^{Int} from the last line in (15), and prove that it is equal to $H_{i,\boldsymbol{a}}\boldsymbol{\delta} + Ab\boldsymbol{e}^i$. First of all, note that addition on boxes is α-complete. Therefore, it is enough to prove that for any matrix B, $\alpha^{\mathsf{Int}}(B\gamma^{\mathsf{Int}}(\boldsymbol{\delta})) = B\boldsymbol{\delta}$. Let $\boldsymbol{\delta}' = \alpha^{\mathsf{Int}}(B\gamma^{\mathsf{Int}}(\boldsymbol{\delta}))$. We have to prove that for each $j \in \{1, \ldots, n\}$, $\delta'_j = B_{j*}\boldsymbol{\delta}$. By definition of α^{Int}, we get

$$\delta'_j = [\inf B_{j*}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}), \sup B_{j*}\gamma^{\mathsf{Int}}(\boldsymbol{\delta})].$$

Note that $\gamma^{\mathsf{Int}}(\boldsymbol{\delta})$ is the Cartesian product $\delta_1 \times \cdots \times \delta_n$, hence $B_{j*}\gamma^{\mathsf{Int}}(\boldsymbol{\delta})$ is a linear combination of intervals. Thus, $B_{j*}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) = B_{j*}\boldsymbol{\delta}$ and this yields $\delta'_j = [\inf B_{j*}\boldsymbol{\delta}, \sup B_{j*}\boldsymbol{\delta}] = B_{j*}\boldsymbol{\delta}$.

It is easy to check that the computational complexity is $O(n^2 + t)$, where t is the time required to compute $H_{i,\boldsymbol{a}}$. We have that $\boldsymbol{a}^T - \boldsymbol{e}^{iT}$ is $O(n)$ and yields a vector of n elements. The result is multiplied by A^{-1}, which requires $O(n^2)$ operations and yields a row matrix of length n. The product $A\boldsymbol{e}^i$ simply selects the i-th column of A, hence requires at most $O(n)$ copy operations. Finally, the product of the column $A\boldsymbol{e}^i$ and the row $(\boldsymbol{a}^T - \boldsymbol{e}^{iT})A^{-1}$ requires $O(n^2)$ operations. At the end, the time required to compute $H_{i,\boldsymbol{a}}$ is $O(n^2)$, which is also the total computational complexity of $\mathsf{assign}^A(i, \boldsymbol{a}, b)$. □

Note that the matrix $H_{i,\boldsymbol{a}}$ may be computed only once for each program point, although this does not change the theoretical complexity of the procedure.

5.4. Forget

We denote with $\mathsf{forget}^A(i)$ the best correct approximation of $\mathsf{forget}(i)$. Intuitively, the effect of an unbounded non-deterministic assignment to the variable x_i is that we lose information on every linear form containing x_i.

Theorem 5 (*Forget*). *Given* $\bot \neq \boldsymbol{\delta} \in \mathsf{Int}$, *we have that*

$$\mathsf{forget}^A(i)(\boldsymbol{\delta}) = \boldsymbol{\delta}',$$

where

$$\delta'_j = \begin{cases} \delta_j & \text{if } A_{ji} = 0, \\ [-\infty, +\infty] & \text{otherwise.} \end{cases}$$

The computational complexity is $O(n)$.

Proof. Note that we may rewrite the concrete forget operator as

$$\begin{aligned}\mathsf{forget}(i)(X) &= \{\boldsymbol{x} + y\boldsymbol{e}^i \mid \boldsymbol{x} \in X, y \in \mathbb{R}\}\\ &= X + \gamma^{\mathsf{Int}}([-\infty, +\infty]\boldsymbol{e}^i). \end{aligned} \tag{16}$$

Accordingly, we may rewrite the abstract forget operator as follows:

$$\begin{aligned}\alpha^A(\mathsf{forget}(i)(\gamma^A(\boldsymbol{\delta}))) &= \alpha^{\mathsf{Int}}(A\,\mathsf{forget}(i)(A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta})))\\ &= \alpha^{\mathsf{Int}}(\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) + A\gamma^{\mathsf{Int}}([-\infty, +\infty]\boldsymbol{e}^i))\\ &= \boldsymbol{\delta} + A([-\infty, +\infty]\boldsymbol{e}^i),\end{aligned}$$

where the last step follows from α-completeness of addition. Note that $\boldsymbol{\delta}' = A([-\infty, +\infty]\boldsymbol{e}^i)$ is an interval vector such that $\delta'_j = 0$ if $A_{ji} = 0$, $\delta'_j = [-\infty, +\infty]$ otherwise. The result follows from the definition of interval sum. □

5.5. Test

We denote by $\mathsf{test}^A(\boldsymbol{a}, b, \bowtie)$ the best correct approximation of $\mathsf{test}(\boldsymbol{a}, b, \bowtie)$. In the following theorem we show that for the abstract test on parallelotopes we may reuse the well known algorithm for abstract test on intervals.

Theorem 6 (*Test*). *Given $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, we have*

$$\mathsf{test}^A(\boldsymbol{a}, b, \bowtie) = \mathsf{test}^{\mathsf{Int}}(A^{-T}\boldsymbol{a}, b, \bowtie),$$

where $A^{-T} \stackrel{\text{def}}{=} (A^{-1})^T$. The computational complexity is $O(n^2)$.

Proof. Given $\boldsymbol{\delta} \in \mathsf{Int}$, we have that

$$\begin{aligned}\alpha^A(\mathsf{test}(\boldsymbol{a}, b, \bowtie)(\gamma^A(\boldsymbol{\delta}))) &= \alpha^{\mathsf{Int}}(A\,\mathsf{test}(\boldsymbol{a}, b, \bowtie)(A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta})))\\ &= \alpha^{\mathsf{Int}}(A(A^{-1}\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \bowtie b\}))\\ &= \alpha^{\mathsf{Int}}(\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \cap \{A\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \bowtie b\})\\ &= \alpha^{\mathsf{Int}}(\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \cap \{\boldsymbol{y} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot (A^{-1}\boldsymbol{y}) \bowtie b\})\\ &= \alpha^{\mathsf{Int}}(\gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \cap \{\boldsymbol{y} \in \mathbb{R}^n \mid ((A^{-T})\boldsymbol{a}) \cdot \boldsymbol{y} \bowtie b\})\\ &= \alpha^{\mathsf{Int}}(\mathsf{test}(A^{-T}\boldsymbol{a}, b, \bowtie)(\gamma^{\mathsf{Int}}(\boldsymbol{\delta})))\\ &= \mathsf{test}^{\mathsf{Int}}(A^{-T}\boldsymbol{a}, b, \bowtie)(\boldsymbol{\delta}).\end{aligned}$$

The complexity of the algorithm for computing $\mathsf{test}^{\mathsf{Int}}$ is $O(n)$. Thus, the complexity for computing $\mathsf{test}^A(\boldsymbol{a}, b, \bowtie)$ is $O(n^2)$, since we need to add the cost for computing $A^{-T}\boldsymbol{a}$. □

Note that $A^{-T}\boldsymbol{a}$ might be computed only once (for each program point) in the analysis, and then subsequent calls to $\mathsf{test}^A(\boldsymbol{a}, b, \bowtie)$ would only cost $O(n)$.

5.6. On the correct implementation of abstract operators

The previous operators on parallelotopes can be correctly implemented under two assumptions:

1. the concrete operations on the target language (the language we want to analyze) are free from rounding errors;
2. the analyzer is able to represent interval vectors and compute abstract operators without rounding errors.

The first assumption means that, if the target language uses floating point arithmetic, abstract operators should be changed accordingly. Miné (2004) shows that a correct algorithm for real numbers may be easily adapted to floating point numbers.

Regarding the second assumption, the simplest way to ensure that the analyzer performs exact computations is using multi-precision rational arithmetic. However, this has a deep impact on performance. Note that, if we were able to compute the exact floating point representation of A^{-1},

Operator	Intervals	Parallelotopes	Octagons	
			w/o closure	with closure
$\cap$	n	n	n^2	n^2
$\cup$	n	n	n^2	n^3
forget	n	n	n	n
assign	n	n^2	n	n^2
test	n	n^2 (n)	n^2 [1]	n^3 [n^2]
widening	n	n	n^2	n^3
narrowing	n	n	n^2	n^3

Fig. 6. Theoretical costs for the basic operators, where n is the number of variables. The value in parenthesis (n) is the cost for subsequent calls, after the first one. The values in square brackets [1] and [n^2] are the costs for the *exact* variant, which only works when conditions are of the kind $\pm x_i \pm x_j \leq c$ or $\pm x_i \leq c$.

we could work entirely with floating point numbers by cleverly choosing the IEEE rounding mode. In the general case, A^{-1} is not representable exactly in floating point arithmetic and, in any case, the algorithm used for inverting the matrix also introduces rounding errors. We may overcome this problem by computing an interval inverse (Rohn, 2010) of A, i.e., a floating point interval matrix Δ such that $A^{-1} \in \Delta$. Simple adaptations of the above algorithms make it possible to use Δ instead of A^{-1} and get correct results.

5.7. Comparing the parallelotope domain

Fig. 6 summarizes and compares the costs of the abstract operators on parallelotopes with the costs of the corresponding operators on the interval and octagon domains. The cost of the test operator on parallelotopes n^2 (n) means that the first call to the test operator is n^2, while subsequent calls cost only n when memoization is used to improve performance.

Comparing parallelotopes with octagons is difficult since some operators in the octagon domain require the constraints to be in closed form, and some operators produce a result which is not in closed form. The cost for computing the closed form is at most $O(n^3)$ for the real and rational cases, but in some cases (after specific operators) an incremental algorithm may be used which only requires $O(n^2)$. Therefore, we have two columns for octagons: the first one shows the cost of the operators without closures, assuming the input argument is closed when needed. The second one shows the cost enforcing closures after each operator. The real cost of the analysis with octagons is in-between, since some closures may be avoided, depending on the order of execution of the abstract operators.

Several different implementations are known for octagon operators, which differ in precision and speed. Here, we provide the costs for the *rel* case, which works for interval linear forms, i.e., linear forms whose coefficients are not constants but intervals. Our algorithms may also be adapted to work with interval linear forms without increasing the theoretical computational cost (see Section 7.4). Only for the test operators, we show in square brackets [1] and [n^2] the costs for the *exact* variant, which only works when conditions are of the kind $\pm x_i \pm x_j \leq c$ and $\pm x_i \leq c$. In this comparison we do not consider the cost of the polyhedral implementation, which is exponential and rarely (if ever) used.

The comparison shows that the parallelotope domain is more costly than the interval domain, but not too much. If we compare it with octagons without closure, some operators are faster with octagons, other with parallelotopes. However, when considering closures, parallelotopes are definitely faster. Finally, we think that parallelotopes may be implemented faster than octagons, especially considering that parallelotope operators are mostly linear algebra operations which are widely studied and for which highly optimized implementations are available.

With respect to the precision of basic operators, all our abstract operators are the best correct approximations of the concrete ones. This also holds for the interval domain but not for the octagon domain.

From the point of view of domain precision, it is immediate to see that, in most cases, parallelotopes are incomparable to both intervals and octagons. If the change of basis matrix is badly chosen, the

analysis may be much less precise than interval analysis. On the other side, parallelotopes may encode constraints which neither interval nor octagons may represent. The choice of the change of basis matrix is therefore crucial to reach a good precision, and this will be the topic of the next section.

6. A statistical tool to choose suitable parallelotopes

The aim of this section is to propose a method to choose the best parallelotope domain for a given program. This amounts to choosing a particular matrix $A \in \mathrm{GL}(n)$.

If we look at partial traces as random variables describing the concrete semantics, we may use multivariate analysis to better understand the correlations in the data. A standard statistical technique consists in finding a suitable linear transformation such that the transformed variables (called *components*) are ordered by decreasing "importance". The criteria of importance that statisticians adopt vary according to the problem. The first approach, from an historical point of view, was to look for components that have the largest variance and are uncorrelated (Pearson, 1901). The two conditions ensure that the loss of information is minimized when the last components are discarded, and that each component can be analyzed separately. The outcome of this approach is what is nowadays called *principal component analysis* (PCA).

Other transformations of the original variables focus on different criteria of importance, at the expense of optimality. In particular, starting from Hausman (1982), the focus moved to having integer or rational coefficients in the components, preserving as much variance and as little correlation as possible. A very nice description of the motivations of this choice can be found in Rousson and Gasser (2004, Introduction). Any technique aiming at having integer or rational coefficients is called *simple component analysis* (SCA).

In the next section we explain how to find the principal components, and we show that using PCA to describe partial traces of programs looks promising (Example 8), but it suffers from a serious problem of instability (Example 10). In Section 6.2 we describe a particular simple component analysis, called OSCA, which in turn solves the instability problem (Example 12).

6.1. PCA

Consider an $m \times n$ real matrix D containing a dataset: each row is a repetition of an experiment and columns are the attributes. Given any length 1 vector $\boldsymbol{v} \in \mathbb{R}^n$, the vector of the orthogonal projections of the dataset D onto the line specified by $\boldsymbol{v}$ is $D\boldsymbol{v}$. The aim of PCA is to find a new basis of length 1 vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$, such that:

1. the variance of $D\boldsymbol{v}_i$ is maximal on the subspace generated by $\boldsymbol{v}_i, \ldots, \boldsymbol{v}_n$;
2. the correlation between $D\boldsymbol{v}_i$ and $D\boldsymbol{v}_j$ is 0 when $i \neq j$.

Consider, for instance, the dataset whose points are depicted in Fig. 7. The points roughly resemble an ellipse, and the first principal component $\boldsymbol{v}_1$ is oriented in the direction of the main axis of the ellipse. The second principal component, which must be orthogonal to the first one, lies on the secondary axis of the ellipse.

It can be proven that conditions 1 and 2 are satisfied by an orthogonal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ of eigenvectors of the $n \times n$ covariance matrix $cov(D)$, chosen in decreasing order of eigenvalue. Recall that the element $c_{i,j}$ in the covariance matrix $cov(D)$ is the covariance between the i-th and j-th columns of D. Since $cov(D)$ is a symmetric matrix, the basis always exists. Geometrically, condition 2 means that the vectors v_i and v_j must be orthogonal. With respect to condition 1, note that maximizing the projection of the variance along a line is equivalent to minimize the squared distance of the data points from the same line. According to Section 5, we denote by A the matrix $(\boldsymbol{v}_1 | \ldots | \boldsymbol{v}_n)^{-1}$, transforming standard coordinates to new coordinates. Note that A is an orthogonal $n \times n$ matrix, that is, $A^{-1} = A^T$.

From the point of view of implementation, principal components might be computed with several techniques, for instance using singular value decomposition.

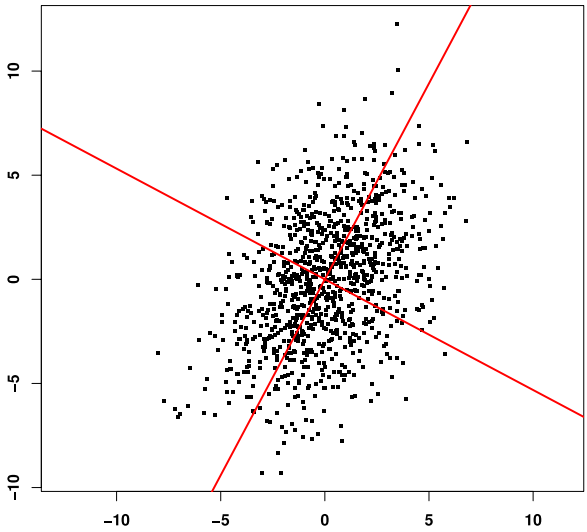

Fig. 7. Points randomly chosen from a bivariate normal distribution and the corresponding principal components.

Example 7. Consider a dataset D with two attributes representing height (in centimeters) and weight (in kilograms) of 8 eleven year old girls (The Open University, 1983):

$$D \stackrel{\text{def}}{=} \begin{bmatrix} 135 & 26 \\ 153 & 55 \\ 154 & 50 \\ 139 & 32 \\ 131 & 25 \\ 149 & 44 \\ 137 & 31 \\ 143 & 36 \end{bmatrix}.$$

The covariance matrix results in the symmetric matrix

$$\text{Cov}(D) = \begin{bmatrix} 73.6964 & 93.4464 \\ 93.4464 & 123.982 \end{bmatrix}.$$

A calculation gives the eigenvectors $\boldsymbol{v}_1 = (0.608350, 0.793669)$ and $\boldsymbol{v}_2 = (-0.793669, 0.608350)$, so that the transformation from the standard to the new coordinate system is given by the matrix

$$A \stackrel{\text{def}}{=} \begin{bmatrix} 0.608350 & 0.793669 \\ -0.793669 & 0.608350 \end{bmatrix}.$$

Fig. 8 shows the eight points in the standard and in the rotated coordinates.

The principal component $\boldsymbol{v}_1$ being $(0.608350, 0.793669)$ means that, to a first approximation, an increase of 1 centimeter in height corresponds to an increase of a little less than 1 kilogram in weight. The vector $\boldsymbol{v}_2$ is what is called a *contrast factor*. Although height and weight are positively correlated, some girls are thinner than others: this difference in the body constitution is represented by the second principal component. □

6.1.1. PCA for static analysis

Example 1 at page 12 can now be explained in terms of PCA.

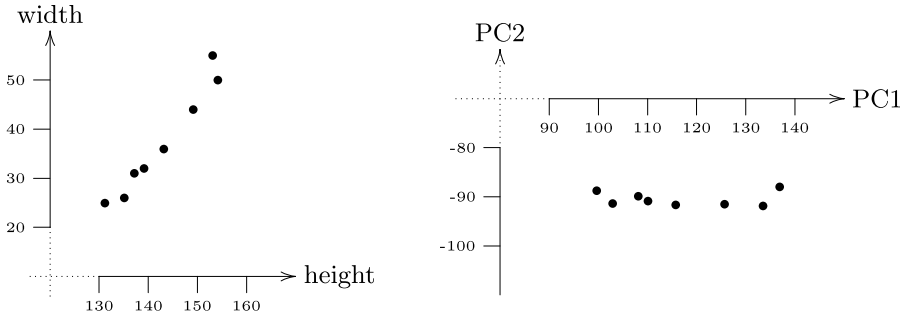

Fig. 8. Points in the girl dataset depicted with standard coordinates (on the left) and rotated coordinates (on the right).

Example 8. Consider the partial execution trace in Fig. 2 at page 4 as a data matrix D. The principal components of D are $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ and $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$, corresponding to the change of basis matrix

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

given in Example 1. □

Note that, for the purposes of static analysis, the most important components are not the ones with maximum variance, but the ones with minimum variance. In the optimal case, when the variance along $\boldsymbol{v}$ is 0, it is very likely that $v_1x_1 + \cdots + v_nx_n = c$ is an invariant of our program (where $x_1, \ldots, x_n$ are the program variables).

Example 9. Consider the partial execution trace in Fig. 2. We have seen that using the matrix A given in Example 8 with the parallelotope domains, we are able to represent the invariants $x + y = 0$ and $x - y \geq 0$ of the program in Fig. 1. However, we may replace the first line of the matrix A with other vectors without loosing accuracy. For example, if we replace the first line of A with $(1, 0)$, we may represent the invariants $x \geq 0$ and $x + y = 0$, which subsume $x - y \geq 0$. However, as soon as we try to replace the second line of A, corresponding to the second principal component, the invariant $x + y = 0$ is no more representable.

Unfortunately, PCA suffers from a serious problem of stability when used in static analysis: small changes in the data cause small changes in the principal components, but these small changes may cause a big loss in precision. This depends on the interaction between the PCA and the parallelotope abstraction function: If X is an unbounded set of points (in $\mathbb{R}^n$), the bounds (in $\bar{\mathbb{R}}$) of the minimum enclosing box of X are not continuous with respect to A, as the following example shows.

Example 10. Consider a small perturbation A_θ of A given by

$$\begin{aligned} A_\theta &\stackrel{\text{def}}{=} \begin{bmatrix} \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} \sqrt{\frac{1}{2}}(\cos\theta - \sin\theta) & -\sqrt{\frac{1}{2}}(\cos\theta + \sin\theta) \\ \sqrt{\frac{1}{2}}(\cos\theta + \sin\theta) & \sqrt{\frac{1}{2}}(\cos\theta - \sin\theta) \end{bmatrix}, \end{aligned}$$

mapping standard coordinates to coordinates with respect to axes rotated clockwise by $45+\theta$ degrees.

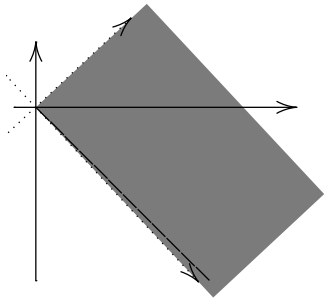

Fig. 9. Bad precision with PCA.

Following Example 1, if we abstract the set X (corresponding to the invariant $x+y=0,\ x-y\geq 0$) in the parallelotope domain associated to A_θ, we get:

$$\begin{aligned}
\alpha^{A_\theta}(X) &= \alpha^{\mathsf{Int}}(A_\theta X) = \alpha^{\mathsf{Int}}(\{A_\theta \boldsymbol{v} \mid \boldsymbol{v} \in X\}) \\
&= \alpha^{\mathsf{Int}}\left(\left\{\begin{bmatrix} \sqrt{\frac{1}{2}}(\cos\theta-\sin\theta) & -\sqrt{\frac{1}{2}}(\cos\theta+\sin\theta) \\ \sqrt{\frac{1}{2}}(\cos\theta+\sin\theta) & \sqrt{\frac{1}{2}}(\cos\theta-\sin\theta) \end{bmatrix}\begin{bmatrix} u \\ -u \end{bmatrix} \mid u\in\mathbb{R}^+\right\}\right) \\
&= \alpha^{\mathsf{Int}}\left(\left\{\begin{bmatrix} \sqrt{2}u\cos\theta \\ \sqrt{2}u\sin\theta \end{bmatrix} \mid u\in\mathbb{R}^+\right\}\right) \\
&= \langle (0,0), (+\infty,+\infty)\rangle,
\end{aligned}$$

where in the last equivalence we have assumed $\theta > 0$.

The concretization $\gamma^{A_\theta}\langle (0,0), (+\infty,+\infty)\rangle$ is $\mathbb{R}^+ \times \mathbb{R}^+$ (the shaded area in Fig. 9). This causes a serious loss of accuracy, for any $\theta > 0$. □

In order to overcome the difficulties outlined in Section 6.1, we need to stabilize the result of the PCA, so that it is less sensible to small changes in the data. Our idea is that, in many cases, we expect that optimal parallelotopes abstracting program states should contain only linear constraints with integer coefficients. This is obvious for programs with integer variables only (such as the program in Fig. 1), or when we are interested in properties described by integer values (such as bounds of arrays, division by 0, etc…). Therefore, we would like to minimally transform the result of the PCA in such a way that A is an integer matrix: thus, we enter the realm of simple component analysis.

6.2. OSCA

Principal components have many theoretical properties, but often statisticians need to interpret them in the context of the data studied. Therefore, they may prefer to replace principal components with other components which are in some way simpler, and therefore easier to interpret. Simplicity means the appearance of some structure in the change of basis matrix, such as groupings of variables, sparseness or integer coefficients. Among the many variants of simple component analysis, we follow the approach of the *orthogonal simple component analysis* (OSCA), introduced by Anaya-Izquierdo et al. (2001).

The authors define a simplification procedure which transforms the orthogonal matrix A given by the PCA into an integer matrix B, given a *threshold* η. More in detail, the integer coefficients of B are chosen as small as possible, subject to the conditions that columns are orthogonal and the angle between the original columns of A and the corresponding columns of B is smaller than η. Note that, in the general case, B is not an orthogonal matrix because its columns are pairwise orthogonal but their length is not one.

The optimal solution of this problem would be too costly from a computational point of view, therefore Anaya-Izquierdo et al. (2001) propose a greedy approach. One principal component $\boldsymbol{v}$ is

chosen, an integer vector $\boldsymbol{v}'$ with coefficients as small as possible and with an angle lesser than η from $\boldsymbol{v}$ is computed and then all the other principal components are projected onto the orthogonal space of $\boldsymbol{v}'$. Finally, the previous steps are repeated recursively on a matrix of smaller dimension then the original one. The order in which principal components are selected is important, since it can give different results.

Another important choice in applying the algorithm is η. Small values of η imply big integer coefficients, while big values of η imply low accuracy. In their original paper Anaya-Izquierdo et al. (2001) show that different values of η give different results all potentially interesting, and propose a procedure which efficiently computes all the different solutions which arise by changing η.

Example 11. Applying OSCA to Example 7, with a threshold $\eta = \pi/8$, replaces $\boldsymbol{v}_1$ with $(1, 1)$ and $\boldsymbol{v}_2$ with $(-1, 1)$. The intuitive interpretation of the principal components given in the original example is preserved, but now it is more evident, while before it was obfuscated by the use of real numbers. □

6.2.1. OSCA for static analysis

Although our aim in simplifying the result of the PCA is different from statisticians' goals, we think that OSCA may be a good candidate for a simplification procedure suited for static analysis. First of all, the result is an integer matrix, and we have already discussed on the appropriateness of this choice. Moreover, the fact that OSCA is biased toward small integers seems well suited for static analysis: it is more likely that linear invariants have coefficients like 1 or 2 than 76484 or 29134. Actually, this is the reason why Octagon works quite well in practice.

In order to apply OSCA, we need to choose a value for η and a selection order for the principal components. By performing several experiments, it seems that a value of $\pi/8$ works quite well in practice (but not always, see the `karr76` example in Section 8). Another possibility would be to perform the analysis with different values of η, using the procedure devised in (Anaya-Izquierdo et al., 2001) to select only the η which gives origin to different results. In this paper, we have decided to work with a fixed η.

With respect to the ordering of principal components, it is important to observe that, since in our case the last principal components are generally more important than the first ones, we start the simplification from the last component and proceed toward the first.

Example 12. Applying OSCA to the matrix A_θ in Example 10, with a perturbation $\theta = 0.1$ and a threshold $\eta = \pi/8$, we get the integer matrix

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.$$

The same holds for any θ between $-\pi/8$ and $\pi/8$. □

7. Optimizations and extensions

In this section we illustrate some improvements over the basic ideas presented in the previous sections.

7.1. Coverage

The PCA matrix is computed on a subset of the variable values excerpted from a finite set of partial execution traces. We use a very basic approach. When analyzing programs without input variables, we simply run the program until it terminates or a threshold on the number of execution steps is reached. In the presence of input variables, the user may supply input values, to be used to generate the partial traces. A possible optimization could be to apply specific coverage techniques, such as dynamic generation of input values or symbolic execution. The aim is to generate input values able to cover every path in the program. For instance, in every `if`-statement, we wish to cover both branches in the set of collected traces. To generate suitable input values, one may collect linear constraints

from test and loop guards, which are solved by using a standard constraint solver (see, e.g., Godefroid et al., 2005; Cadar et al., 2006; Tillmann and de Halleux, 2008). We believe that all these methods may considerably help in achieving a high code coverage, and may be used to improve the precision of the overall analysis.

A different probabilistic technique recently proposed in Gulwani and Necula (2003, 2005) may also be used. The aim is to collect information on all the possible execution traces with a single execution trace, by performing a single run through the program. The basic idea is to relax the semantics of branching points (such as *if–then–else*) by combining the values of both paths. For instance, one could take 40% of one path and 60% of the other. For any variable x involved in the statement, if x_1 and x_2 are, respectively, the values computed on the two paths, we use a combined value $0.4 * x_1 + 0.6 * x_2$. By randomly choosing the weights in each branch, the effect is to obtain a trace which involves all the possible traces. The authors show that this strategy captures most of the affine relationships among variables of the program. Of course, the traces obtained may fail to be a real run of the program, but they may be very close. Our approach may largely benefit from this technique, since we do not require correctness in collecting execution traces, and we may as well accept traces which are not a real run of the program.

7.2. Partitioning the set of variables according to usage pattern

The matrix computed by the PCA may combine variables which are not logically related. For instance, when dealing with arrays, we can recognize index variables, used to identify positions in arrays, and data variables, used to temporary store data from arrays (to swap elements, to compute new elements, etc.). We have experienced that not combining these variables in the PCA may considerably enhance the overall precision of the analysis. This suggests, as a possible improvement, that one could partition the set of variables in two sets and apply the PCA separately on the two sets. The partitioning may be determined statically before applying the PCA, by syntactically examining the program source code. Initially, all the variables are in the index-variable sets (and the data-variable set is empty). Whenever we found a variable assigned to/from an array element, then we move that variable from the index-variable set to the data-variable set. We do the same for guards involving array elements.

This idea may be refined by computing finer partitions, each one composed of variables which interact during the execution of the program. This is what Guo et al. (2006) call *abstract types*. Abstract types may be computed either at run-time or at compile-time, with tools such as Ajax (O'Callahan, 2001) and Lackwit (O'Callahan and Jackson, 1997).

In addition, we may partition the program code (for example around loops), perform a different PCA on each partition, and change the abstract domain appropriately when crossing partitions. In the extreme, we could choose different parameters for each program point, like Sankaranarayanan et al. (2005) do for template polyhedra.

7.3. Packing

The optimization discussed in Section 7.2 is very similar to packing, used in the ASTREÉ analyzer (Blanchet et al., 2003) with the octagon domain. The main difference is that we produce partitions of the set of variables, while packs generally have non-empty intersections. Moreover, while we use partitioning to improve the precision and speed of the analysis, packing is used exclusively to increase speed (and scalability), while reducing accuracy.

Note that standard packing may be used with the parallelotope domains, and this may improve precision with respect to a non-packed analysis. This is because when packing is used in Octagon, it reduces the number of linear forms encoded in the domain. In the case of parallelotopes, however, non-disjoint packs may increase the number of linear forms encoded. For example, when we analyze a program with n variables without packing, the parallelotope domain encodes n linear constraints. But if we use two packs of $2n/3$ variables each, then $4n/3$ linear forms may be possibly encoded.

7.4. Interval linear forms

Sometimes, a more general form of assignment is desired, where $\boldsymbol{a}$ is an interval vector and b is an interval. This may be used to model non-determinism (such as the input from an external sensor) or to handle non-linear expressions by a preliminary linearization process (Miné, 2006). In this case, given $\boldsymbol{a} \in \mathbb{I}^n$ and $b \in \mathbb{I}$, we define $\mathsf{assign}(i, \boldsymbol{a}, b) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ as follows:

$$\mathsf{assign}(i, \boldsymbol{a}, b)(X) \stackrel{\text{def}}{=} \{\mathsf{assign}(i, \boldsymbol{a}', b')(\boldsymbol{x}) \mid \boldsymbol{x} \in X, \boldsymbol{a}' \in \mathbb{R}^n, b' \in \mathbb{R}, \underline{\boldsymbol{a}} \leq \boldsymbol{a}' \leq \overline{\boldsymbol{a}},\ \underline{b} \leq b' \leq \overline{b}\}. \tag{17}$$

The abstract operators in the interval and parallelotope domains are essentially the same as in the standard case, provided that $\boldsymbol{a}$ is now an interval vector, b is an interval and $H_{i,\boldsymbol{a}}$ is an interval matrix.

7.5. The case of integer variables

Although we have presented our domain as an abstraction of $\wp(\mathbb{R}^n)$, we may apply the same construction to build an abstraction of $\wp(\mathbb{Z}^n)$, in order to analyze programs with integer variables. In this case, whenever A is an integer matrix, we may perform almost all the computations on integers. In fact, observe that A^{-1} is an integer matrix divided by an integer number $d \in \mathbb{Z}$. Since all the operations involved in assign^A are linear, d may be factored out and applied at the end of the computation only, before rounding the intervals to integer bounds.

Let $\mathsf{Int}_{\mathbb{Z}}$ be the set of boxes with integers (and infinite) bounds. We define a concretization map $\gamma_{\mathbb{Z}}^{\mathsf{Int}} : \mathsf{Int} \to \wp(\mathbb{Z}^n)$ by $\gamma_{\mathbb{Z}}^{\mathsf{Int}}(\boldsymbol{\delta}) \stackrel{\text{def}}{=} \gamma^{\mathsf{Int}}(\boldsymbol{\delta}) \cap \mathbb{Z}^n$. Analogously, for a given change of basis matrix A, we define $\gamma_{\mathbb{Z}}^{A} : \mathsf{Int} \to \wp(\mathbb{Z}^n)$ by $\gamma_{\mathbb{Z}}^{A}(\boldsymbol{\delta}) \stackrel{\text{def}}{=} \gamma^{A}(\boldsymbol{\delta}) \cap \mathbb{Z}^n$. Since $\gamma_{\mathbb{Z}}^{\mathsf{Int}}$ and $\gamma_{\mathbb{Z}}^{A}$ are co-continuous, the corresponding $\alpha_{\mathbb{Z}}^{\mathsf{Int}}$ and α_{Z}^{A} can be defined.

The first problem is that, while $\langle \alpha_{\mathbb{Z}}^{\mathsf{Int}}, \gamma_{\mathbb{Z}}^{\mathsf{Int}} \rangle$ is a Galois insertion, $\langle \alpha_{\mathbb{Z}}^{A}, \gamma_{\mathbb{Z}}^{A} \rangle$ is only a Galois connection, since there are different integer boxes which contain exactly the same set of integer points.

Example 13. The set of equations $\{x - y = 0, x + y = 1\}$ have no integer solutions. This means that, given the matrix $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$, the box $\langle (0, 1), (0, 1) \rangle$ has an empty concretization, just as the empty box $\bot$. □

In other domains, such as Octagon (Miné, 2006), it is common to normalize the abstract objects in such a way that all the inequalities it encodes are saturated. In our case the normalization process seems to be costly. Therefore, we prefer to work with non-normalized objects, even if we may incur in some loss of precision.

We show now that, if A is an integer matrix, all abstract operators in the parallelotope domains may be performed with integer arithmetic. The cases of union, intersection and forget operators are trivial. For the assignment, the problem is that we need to compute the matrix $H_{i,\boldsymbol{a}} = A^{-1} Z_{i,\boldsymbol{a}} A$, and A^{-1} may be non-integer, even if A is.

This seems to require the use of rational or floating point arithmetic. However, $A^{-1} \det A = A^*$, where A^* is the adjoint of A and is an integer matrix. Therefore, given $H^*_{i,\boldsymbol{a}} = A^* Z_{i,\boldsymbol{a}} A$, we may compute $H^*_{i,\boldsymbol{a}} \boldsymbol{\delta} + \det(A) A b \boldsymbol{e}^i$ with integer interval arithmetic, and divide by $\det(A)$ at the end. Non-integer numbers may be replaced by their upper integral parts (for lower bounds) or lower integral parts (for upper bounds).

For the test operator, we know that $\mathsf{test}^A(\boldsymbol{a}, b, \bowtie) = \mathsf{test}^{\mathsf{Int}}(A^{-T}\boldsymbol{a}, b, \bowtie)$. However, it is equivalent to $\mathsf{test}^{\mathsf{Int}}((A^*)^T \boldsymbol{a}, b \det A, \bowtie)$ which only requires integer numbers.

Note that other optimizations are possible for integer numbers, not related to the fact that A is integer. For example, $\mathsf{test}(\boldsymbol{a}, b, <)$ may be replaced with $\mathsf{test}(\boldsymbol{a}, b-1, \leq)$ which gives better precision.

7.6. Combination with range analysis

Using the parallelotope domains, we have occasionally experienced some problems in the bootstrap phase of the analysis. Consider the sample program `start1` in Fig. 10. If we perform the

```
start1 = function()
{
  x=10
  y=x
}
```

```
start2 = function(x)
{
  y=10
  x=y
}
```

Fig. 10. Example programs.

analysis with the interval domain, we may easily infer that, at the end of the function, both variables x and y assume the value 10. However, using the parallelotope domain with the axes rotated clockwise by 45°, the analysis starts with the abstract state which covers the whole of $\mathbb{R}^2$. The assignment $x = 10$ has no effect: since there are no bounds on the possible values for y, then nothing may be said about $x + y$ and $x - y$, even if we know the value of x. Therefore, after the second assignment, we only know that $x - y = 0$, losing precision with respect to range analysis, although $x = y = 10$ may be expressed in the rotated domain as $x + y = 20$, $x - y = 0$.

The problem arises from the fact that assignments are naturally biased toward the standard axes, since the left hand side is always a variable. At the beginning of the analysis, when the abstract state does not contain any constraint, all constant assignments are lost, and this is generally unfavorable to the precision of the analysis.

It is possible to overcome this problem by initializing all the local variables to zero, as done in many programming languages, or by using the simultaneous assignment operator. Unfortunately, these methods do not always solve all the problems, due to the presence of input parameters. Consider the program `start2` in Fig. 10. In this case, we assume that $y = 0$ at the beginning of the function, but we cannot assume that $x = 0$, since this is a parameter. However, our parallelotopes (the 45-degree clockwise rotated boxes) cannot express the fact that $y = 0$. Hence the abstract state at the beginning of the function is the full space $\mathbb{R}^2$, and the result at the end of the function is again $x - y = 0$. From the point of view of precision, an optimal solution to this kind of problem would be to use the reduced product of the interval domain and parallelotope domains. However, this may severely degrade performance. A good trade-off could be to perform both analyses in parallel: at the end of each abstract operator, we use the information from one domain to refine the other, and vice versa. Given a box and a parallelotope, a satisfactory and computationally affordable solution is to compute the smallest parallelotope which contains the box, and then the intersection between the two parallelotopes. The symmetric process can be used to refine the box. We have adopted this solution in our analyzer (see Feret, 2004, 2005 for a similar approach).

7.7. *Fixing constraints a priori*

A limitation of the parallelotope domains is that the number of different linear forms simultaneously handled is limited to n, where n is the number of variables. When the change of basis matrix is computed automatically by the statistical tools, it is possible that the chosen linear forms are not very useful for the intended application.

For example, consider the following program,[1] where • stands for non-deterministic test.

```
x = 0
y = 0
while ( • ) {
  y = y+1
  if ( • )
    x = 2*y
  else
    x = -2*y
}
```

[1] This program has been suggested by one of the referees.

If we collect the program trace at the entry point of the while loop, we get a sequence where y increases and x jumps between $-2y$ and $2y$. The first principal component will be very close to the y-axis, which is in the middle between the lines $x = 2y$ and $x = -2y$. The second component is then forced to be the x-axis. Thus, the parallelotope domain turns out to be equivalent to the interval domain, and the analysis cannot prove the invariant $-2y \leq x \leq 2y$. However, if we manually feed the parallelotope domain with the change of basis matrix $A = \begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix}$, we are able to represent constraints on the linear forms $x + 2y$ and $x - 2y$, and to prove the above invariant.

As a further example, if the analysis aims to check the absence of divisions by zero, it is generally better to include in the change of basis matrix all the linear forms which appear as divisors in the program. Obviously, this is only possible if these linear forms are linearly independent, otherwise the matrix would be singular. When there are fewer linear forms than variables, the problem is how to choose the remaining linear forms.

A possible way to proceed is to collect partial traces as in the standard approach, project these traces onto the space orthogonal to the chosen linear forms, and process the result with OSCA or other statistical tools. In this way, we obtain other linear forms which are by construction linearly independent of the chosen ones, and luckily the best choice for generating and propagating invariants.

8. Experimental evaluation

We have implemented a prototype for the intra-procedural analysis of a simple imperative language, to investigate the feasibility of the ideas introduced above (Amato et al., 2010b). The prototype has been written in R, a language and environment for statistical computing (R Development Core Team, 2009). We analyze programs written in an imperative fragment of the *R* language, which includes assignments, conditionals and loops. The analyzer instruments the program under analysis to record the values of the variables at every program point, recovers the partial execution traces starting from the input values (provided by the user), computes the orthogonal simple components, and finally performs the static analysis. The analysis may be performed with either the interval domain, the parallelotope domains, or with their combination. Program equations are solved with a *recursive chaotic iteration strategy* on the *weak topological ordering* induced by the program structure (see Bourdoncle, 1993). The analyzer uses the standard widening (Cousot and Cousot, 1976) which extrapolates unstable bounds to infinity and the standard narrowing which improves infinite bounds only. Correctness of abstract operators is ensured using rational arithmetic.

The main drawback of R, at least for our application, is speed. For a prototype, this was deemed less important than fast coding. However, this means that we cannot compare the effective speed of the parallelotope domains with the speed of octagons or polyhedra, because all the standard implementations of the latter domains, in libraries such as `APRON` (Jeannet and Miné, 2009) or `PPL` (Bagnara et al., 2008), are in C or C + +.

Although an exhaustive comparison of the speed and precision of the domains of parallelotopes with other domains is outside the scope of this paper, we present here some preliminary results. We have tested the analyzer with some simple programs collected from the literature. In Fig. 13 we show the results for the following programs: `xyline`—the example program in Fig. 1; `cousot78`—the program in Fig. 11, which is an instance of a skeletal program in Cousot and Halbwachs (1978); `karr76` – the program in Fig. 12, which appeared in Karr (1976); `bsearch` – binary search over 100-element arrays, as appeared in Cousot and Cousot (1976); `bsort` – bubblesort over 100-element arrays, which is the first example program in Cousot and Halbwachs (1978); `heapsort` – which implements a standard heapsort algorithm and `merge` – merge of two ordered arrays. All programs have at least one loop. For each program, we show the abstract state inferred by the analyzer at the beginning of the loop. Since `bsort` and `heapsort` have two nested loops, we only show the abstract state for the outer one. For the program `merge` we show the abstract state only for the first of its loops.

For the parallelotope and combined domains, we have used a change of basis matrix determined by the orthogonal simple component analysis with an accuracy threshold of $\frac{\pi}{8}$. The only exception is `karr76`, where we have used an accuracy of $\frac{\pi}{32}$, since OSCA could not find a good matrix using $\frac{\pi}{8}$. In the examples marked with $*$ we have partitioned the set of variables, according to Section 7.2.

```
cousot78 = function()
{
  i = 2
  j = 0
  while (TRUE) {
    if (i*i==4)
      i = i+4
    else {
      j = j+1
      i = i+2
} } }
```

Fig. 11. An instance of an example in Cousot and Halbwachs (1978).

```
karr76 = function(k)
{
     i = 2
     j = k+5
     while (TRUE) {
            i = i+1
            j = j+3
     }
}
```

Fig. 12. An example program in Karr (1976).

program	Intervals	Parallelotopes	combined	Octagon
xyline		$(0 \le x)$ $(y \le 0)$ $-x + y \le 0$ $x + y = 0$	as ptope	as ptope
cousot78	$2 \le i$ $0 \le j$ $(2 \le i + j)$	$(2 \le i)$ $(-i + j \le -2)$ $4 \le 2i + j$ $\mathbf{-i + 2j \le -2}$	$2 \le i$ $0 \le j$ $(2 \le i + j)$ $(-i + j \le -2)$ $4 \le 2i + j$ $\mathbf{-i + 2j \le -2}$	$2 \le i$ $0 \le j$ $2 \le i + j$ $-i + j \le -2$ $(4 \le 2i + j)$
karr76 $(\eta = \pi/32)$	$2 \le i$		$2 \le i$ $(5 \le j - k)$ $\mathbf{3i - j + k = 1}$	$2 \le i$ $5 \le j - k$
bsearch	$1 \le lwb \le 100$ $1 \le upb \le 100$ $0 \le m \le 100$ $(-99 \le upb - lwb \le 99)$ $(-100 \le m - lwb \le 99)$	$0 \le upb - lwb$	as int+ptope	$1 \le lwb \le 100$ $1 \le upb \le 100$ $0 \le m \le 100$ $0 \le upb - lwb \le 99$ $\mathbf{-99} \le m - lwb \le 99$
bsearch*	$1 \le lwb \le 100$ $1 \le upb \le 100$ $0 \le m \le 100$ $(-99 \le upb - lwb \le 99)$ $(-100 \le m - lwb \le 99)$ $(-200 \le -lwb - upb + 2m \le 198)$	$\mathbf{0} \le upb - lwb \le 99$ $(\mathbf{-101/2} \le m - lwb \le \mathbf{299/4})$ $\mathbf{-101} \le -upb - lwb + 2m \le \mathbf{101/2}$	as int + ptope	$1 \le lwb \le 100$ $1 \le upb \le 100$ $0 \le m \le 100$ $0 \le upb - lwb \le 99$ $-99 \le m - lwb \le 99$ $(-199 \le -upb - lwb + 2m \le 198)$
bsort	$1 \le b$ $0 \le j, t$ $(0 \le j + t)$	$1 \le b$	as int + ptope	$1 \le b$ $0 \le j, t$ $0 \le j + t$ $\mathbf{0 \le b - t \le 100}$ $\mathbf{b - j \le 100}$ $\mathbf{0 \le j - t}$
bsort*	as above	as above	$1 \le b \le \mathbf{100}$ $0 \le j \le \mathbf{100}$ $0 \le t \le \mathbf{99}$ $0 \le j + t \le \mathbf{199}$ $0 \le j - t \le \mathbf{100}$ $(-98 \le b - t \le 100)$ $(\mathbf{-99} \le b - j \le 100)$	$1 \le b$ $0 \le j$ $0 \le t$ $0 \le j + t$ $0 \le j - t$ $\mathbf{0} \le b - t \le 100$ $b - j \le 100$
heapsort*	$2 \le n, r$ $1 \le l$	$-n + r \le 0$	as int + ptope	$2 \le n, r$ $1 \le l$ $-n + r \le 0$
merge*	$1 \le i, j, k, n, m$ $(3 \le i + j + k)$	$(1 \le k)$ $3 \le i + j + k$ $\mathbf{-i - j + k = -1}$	as int + ptope	$1 \le i, j, k, n, m$ $(3 \le i + j + k)$ $\mathbf{0 \le -i + k, -j + k, -j + m, -i + n}$

Fig. 13. Results of the analyses for several programs and domains. Constraints in parentheses are not part of the result of the analyses, but may be inferred from them.

When comparing intervals and parallelotopes, we find that the results in the parallelotope domain are more precise for `xyline`, the results in the interval domain are more precise for `karr76` and `bsort`, while, in all the other cases, they are incomparable.

The combined domain is obviously more precise than both intervals and parallelotopes and it can also improve over the results obtained by the two separate analyses, as in `bsort`*, where the combined domain is able to prove that all accesses to arrays are correct, and in `karr76`, where we find the loop invariant $3i - j + k = 1$.

In order to compare the result of the combined domain to the Octagon domain (Miné, 2006), we have used the Interproc analyzer (Jeannet, 2004; Jeannet and Miné, 2009) with standard parameters. This has required converting the sample programs from the R syntax to the syntax supported by Interproc. In Fig. 13, the results for Octagon are simplified by removing constraints which are implied by other constraints.

In the general case, the combined domain and Octagon are incomparable. For instance, with `cousot78`, `karr76`, `bsearch`* and `bsort`*, we obtain more precise results with the combined domain, while for `bsearch` and `bsort`, Octagon is more precise (but the theoretical complexity of its operators is greater). In particular, for `cousot78` we were able to obtain the property $-i + 2j \leq -2$ which cannot be represented in Octagon, and cannot be inferred by the corresponding results. It is worth noting that, for `bsort`*, the analysis on the combined domain gives more precise results than Octagon, even if it uses octagonal constraints only. This is due to the fact that some operators in Octagon are not the best correct abstractions.

A practical comparison of speed between our implementation and Octagon is not possible at the moment, since our implementation in R is definitively slower than the APRON (Jeannet and Miné, 2009) library used in Interproc. However, the theoretical costs in Fig. 6 show that most parallelotope operators are definitely faster than Octagon's.

9. Related work

The approach we propose in this paper is to use parametric numerical abstract domains, and determine the best parameters for a given program using statistical analysis of partial traces.

The phase of statistical analysis may be considered as a way to dynamically infer invariants, although it only determines the shape (that is, the linear combination of variables) of the invariants and not the real bounds. The latter are determined and guaranteed to be true by the static analysis phase. Several works try to determine invariants dynamically. Among them we recall the Daikon invariant generator (Ernst et al., 2001) and the approach based on random interpretation (Gulwani and Necula, 2005). Daikon executes the program, collects variable values at different program points, and checks the (likely) validity of a set of predefined invariants. The random interpretation approach collects probabilistic execution traces, in order to directly derive linear relationships between program variables, which hold with a given probability. We cannot immediately replace SCA with these tools: the parallelotope domains need a basis for the vector space of variable values, while these approaches generate a set of linear combinations of variables which, in the general case, is neither linear independent nor a generator.

The abstract domain of parallelotopes is a weakly relational abstract domain such as octagons (Miné, 2006), weighted hexagons (Fulara et al., 2010), two variables per inequality (Simon et al., 2003) and many others. From the point of view of precision, it is incomparable with all the other weakly relational abstract domains and even with the interval domain. In practice, the precision of the analysis with parallelotopes strictly depends on the choice of the change of basis matrix. Most of the time, parallelotope analysis should be performed in lockstep with interval analysis to reach a good level of precision. From the point of view of performance, parallelotopes are a bit slower than interval but faster than other weakly relational abstract domains.

The idea of parameterizing analyses for a class of programs or for a single program has been pursued in many papers. The analysis for digital filters proposed in Feret (2004) is an example of domains developed for a specific class of applications. The same holds for the domain of arithmetic–geometric progressions (Feret, 2005), used to determine restrictions on the value of variables, as a function of the program execution time.

Other domains have parameters which may be tuned for a given program: it is the case for the domain of symbolic intervals (Sankaranarayanan et al., 2007), which depends on a total ordering of variables, and most importantly, for the domain of template polyhedra (Sankaranarayanan et al., 2005). In the latter, the authors fix *a priori*, for each program point, a matrix A and consider all the polyhedra of type $A\boldsymbol{x} \leq \boldsymbol{b}$. The choice of A is what differentiates template polyhedra from other domains, where the matrix either is fixed for all the programs (such as intervals or octagons) or varies freely (such as polyhedra). A limit of the parallelotope domains is that the number of constraints it can represent is bounded by n, and they must be linearly independent. This differs from polyhedral templates which allow an unlimited number of constraints. However, it is this limitation which guarantees the existence of very fast basic operators, without the need to resort to linear programming methods.

There are also parametrization strategies applicable to almost all numeric domains. Many of them are discussed in Blanchet et al. (2003). For example, the accuracy of widening operators can be enhanced through the adoption of intermediate thresholds, recoverable from a simple syntactic analysis of the program (e.g., maximum size of arrays and declared constants). Moreover, the complexity of relational analyses can be reduced by using *packing* (see Section 7.3). These strategies are orthogonal to our approach, and can be applied to our domains as well.

In all the cases shown above, parameters are chosen after a syntactic inspection of the program, while in our case we perform a dynamic analysis of the program's behavior. To the best of our knowledge, the only work which combines static and dynamic analysis is the paper of Gupta et al. (2009), whose ideas have been implemented in `InvGen` (Gupta and Rybalchenko, 2009). In this work, dynamic analysis is used to improve the efficiency of a constraint-based invariant generator. Programs are executed in order to collect variable values at different program points, just like in our approach. These values are used to generate new linear constraints on the invariant's parameters, which help to reduce the search space of the constraint solver. Note that `InvGen` also gives the opportunity to perform a symbolic execution of the program, instead of a real execution.

Another paper which has strong similarity to ours is the work of Stursberg and Krogh (2003) on approximating the reachable states for hybrid systems. In order to represent concisely the set of points reachable by the system during a time step ΔT, they use parallelotopes,[2] while PCA finds out a good orientation for the axis. The idea is very similar to ours, but they do not provide any information on the implementation of the abstract operators. Moreover, their work is not directly applicable to static analysis of software since they only consider bounded parallelotopes. This leads to ignore the problems which brought us to choose SCA instead of PCA.

10. Conclusions and future work

We have presented a new technique for shaping numerical abstract domains to single programs, by applying a "best" linear transformation to the space of variable values. One of the main advantages of this technique is the ability to transform non-relational analysis into relational ones, by choosing the abstract domain which best fits for a single program. Moreover, this idea may be immediately applied to any numerical abstract domain which is not closed by linear transformations, such as octagons (Miné, 2006), bounded differences (Miné, 2001), simple congruences (Granger, 1989). For the transformed domain, it suffices to give specialized algorithms for the assignment and forget operators, since the abstract operators for union, intersection and test are derived immediately from the original operators.

We have built a prototype analyzer and, as an application, we have fully developed our technique for the interval domain. The experimental evaluation seems promising, but also shows that there is still space for many improvements, most of which have been discussed in Section 7.

The use of linear transformations also suggests the combination of PCA with different approaches. We may infer the axes in the new coordinate system from both the semantics and the syntax of the program. The analysis could vastly benefit from the ability to express constraints occurring in the

[2] which they call *oriented rectangular hulls*.

linear expressions of the program, especially in loop guards and array accesses. However, the syntactic approach alone is not recommended, since not all the interesting invariants appear as expressions in the source code. For example, the `cousot78` program in Fig. 11 does not contain the expressions `i+j`, `j-i` or `2*j-i`: nonetheless, the analysis was able to prove invariants on these constraints (see Fig. 13). To overcome this limitation, we may use the probabilistic invariants found by the analysis in Gulwani and Necula (2005) instead of using the syntax of the program.

Writing the implementation in R has been useful for rapid prototyping, but porting the code to a faster programming language, possibly within the framework of well known libraries such as APRON (Jeannet and Miné, 2009) or PPL (Bagnara et al., 2008), would make it available to a wider community, while improving performance.

Finally, remark that we never use the fact that the axes obtained by OSCA are orthogonal. This suggests that a different SCA might be used to obtain the matrix A. In particular, one could use the SCA described in Rousson and Gasser (2004), which is not related to PCA.

References

Amato, G., Parton, M., Scozzari, F., 2010a. Deriving numerical abstract domains via principal component analysis. In: Cousot, R., Martel, M. (Eds.), 17th International Symposium, SAS 2010, Perpignan, France, September 14–16, 2010, Proceedings. In: Lecture Notes in Computer Science. Springer, vol. 6337. Berlin Heidelberg, pp. 134–150.

Amato, G., Parton, M., Scozzari, F., 2010b. A tool which mines partial execution traces to improve static analysis. In: Barringer, H., et al. (Eds.), First International Conference, RV 2010, St. Julians, Malta, November 1–4, 2010. Proceedings. In: Lecture Notes in Computer Science., vol. 6418. Springer, Berlin Heidelberg, pp. 475–479.

Anaya-Izquierdo, K., Critchley, F., Vines, K., March 2011. Orthogonal simple component analysis: a new, exploratory approach. Annals of Applied Statistics 5 (1), 486–522.

Bagnara, R., Hill, P. M., Zaffanella, E., 2008. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72 (1–2), 3–21.

Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X., June 7–14 2003. A static analyzer for large safety-critical software. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. PLDI'03. ACM Press, San Diego, California, USA, pp. 196–207.

Bourdoncle, F., 1993. Efficient chaotic iteration strategies with widenings. In: Bjørner, D., Broy, M., Pottosin, I.V. (Eds.), Formal Methods in Programming and Their Applications, International Conference Academgorodok, Novosibirsk, Russia June 28–July 2, 1993 Proceedings. In: Lecture Notes in Computer Science, vol. 735. Springer, Berlin Heidelberg, pp. 128–141.

Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., Engler, D. R., 2006. Exe: automatically generating inputs of death. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. CCS'06. ACM, New York, NY, USA, pp. 322–335.

Chang, B.-Y.E., Rival, X., 2008. Relational inductive shape analysis. In: Principles Of Programming Languages. POPL'08. In: SIGPLAN Not., vol. 43(1). ACM, New York, NY, USA, pp. 247–260.

Colóon, M.A., Sipma, H.B., 2001. Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001 Proceedings. In: Lecture Notes in Computer Science, vol. 2031. Springer, Berlin Heidelberg, pp. 67–81.

Cousot, P., 1999. The calculational design of a generic abstract interpreter. In: Broy, M., Steinbrüggen, R. (Eds.), Calculational System Design. NATO Science Series F. IOS Press, Amsterdam, pp. 421–506.

Cousot, P., Cousot, R., 1976. Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming. Dunod, Paris, France, pp. 106–130.

Cousot, P., Cousot, R., Jan. 1979. Systematic design of program analysis frameworks. In: POPL'79: Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM Press, New York, NY, USA, pp. 269–282.

Cousot, P., Cousot, R., Jul. 1992. Abstract interpretation and applications to logic programs. The Journal of Logic Programming 13 (2–3), 103–179.

Cousot, P., Halbwachs, N., Jan. 1978. Automatic discovery of linear restraints among variables of a program. In: POPL'78: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM Press, New York, NY, USA, pp. 84–97.

Dor, N., Rodeh, M., Sagiv, M., 2001. Cleanness checking of string manipulations in C programs via integer analysis. In: Cousot, P. (Ed.), Static Analysis, 8th International Symposium, SAS 2001 Paris, France, July 16–18, 2001 Proceedings. In: Lecture Notes in Computer Science, vol. 2126. Springer, Berlin Heidelberg, pp. 194–212.

Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D., 2001. Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering 27 (2), 99–123.

Feret, J., 2004. Static analysis of digital filters. In: Schmidt. pp. 33–48.

Feret, J., 2005. The arithmetic-geometric progression abstract domain. In: Cousot, R. (Ed.), Verification, Model Checking, and Abstract Interpretation – 6th International Conference, VMCAI 2005, Paris, France, January 17–19, 2005. Proceedings. In: Lecture Notes in Computer Science, vol. 3385. Springer, pp. 42–58.

Fulara, J., Durnoga, K., Jakubczyk, K., Schubert, A., October 2010. Relational abstract domain of weighted hexagons. Electronic Notes in Theoretical Computer Science 267, 59–72. http://dx.doi.org/10.1016/j.entcs.2010.09.006.

Godefroid, P., Klarlund, N., Sen, K., 2005. Dart: directed automated random testing. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI'05. ACM, New York, NY, USA, pp. 213–223.

Granger, P., 1989. Static analysis of arithmetical congruences. International Journal of Computer Mathematics 32.

Gulavani, B.S., Gulwani, S., 2008. A numerical abstract domain based on *expression abstraction* and *max operator* with application in timing analysis. In: Gupta, A., Malik, S. (Eds.), Computer Aided Verification, 20th International Conference, CAV 2008 Princeton, NJ, USA, July 7–14, 2008 Proceedings. In: Lecture Notes in Computer Science, vol. 5123. Springer, Berlin Heidelberg, pp. 370–384.

Gulwani, S., Necula, G. C., 2003. Discovering affine equalities using random interpretation. In: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. POPL'03. ACM, New York, NY, USA, pp. 74–84. http://doi.acm.org/10.1145/604131.604138.

Gulwani, S., Necula, G. C., 2005. Precise interprocedural analysis using random interpretation. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL'05. ACM, New York, NY, USA, pp. 324–337. http://doi.acm.org/10.1145/1040305.1040332.

Guo, P.J., Perkins, J.H., McCamant, S., Ernst, M.D., 2006. Dynamic inference of abstract types. In: Proceedings of the 2006 International Symposium on Software Testing and Analysis. ISSTA'06. ACM, New York, NY, USA, pp. 255–265. http://doi.acm.org/10.1145/1146238.1146268.

Gupta, A., Majumdar, R., Rybalchenko, A., 2009. From tests to proofs. In: Kowalewski, S., Philippou, A. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems 15th International Conference, TACAS 2009, York, UK, March 22–29, 2009. Proceedings. In: Lecture Notes in Computer Science, vol. 5505. Springer, Berlin Heidelberg, pp. 262–276.

Gupta, A., Rybalchenko, A., 2009. InvGen: an efficient invariant generator. In: Bouajjani and Maler. pp. 634–640.

Hausman Jr, R.E., 1982. Constrained multivariate analysis. In: Optimization in statistics. In: Stud. Management Sci., vol. 19. North-Holland, Amsterdam, pp. 137–151.

Hickey, T.J., Ju, Q., van Emden, M.H., 2001. Interval arithmetic: from principles to implementation. Journal of the ACM 48, 1038–1068.

Jeannet, B., 2004. Interproc Analyzer for Recursive Programs with Numerical Variables. INRIA, software and documentation are available at the following URL: http://pop-art.inrialpes.fr/interproc/interprocweb.cgi. Last accessed: 2010-06-11.

Jeannet, B., Miné, A., 2009. APRON: a library of numerical abstract domains for static analysis. In: Bouajjani and Maler. pp. 661–667.

Karr, M., 1976. Affine relationships among variables of a program. Acta Information 6, 133–151.

Kearfott, R. B., 1996. Interval computations: Introduction, uses, and resources. Euromath Bulletin 2, 95–112.

Miné, A., 2001. A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (Eds.), Programs as Data Objects, Second Symposium, PADO2001 Aarhus, Denmark, May 21–23, 2001 Proceeding. In: Lecture Notes in Computer Science, vol. 2053. Springer, Berlin Heidelberg, pp. 155–172.

Miné, A., 2004. Relational abstract domains for the detection of floating-point run-time errors. In: Schmidt. pp. 3–17.

Miné, A., 2004. Weakly relational numerical abstract domains. Ph.D. Thesis, École Polytechnique.

Miné, A., Mar. 2006. The octagon abstract domain. Higher-Order and Symbolic Computation 19 (1), 31–100. http://www.di.ens.fr/~mine/publi/article-mine-HOSC06.pdf.

Miné, A., 2006. Symbolic methods to enhance the precision of numerical abstract domains. In: Emerson, E.A., Namjoshi, K.S. (Eds.), Verification, Model Checking, and Abstract Interpretation. 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006. Proceedings. In: Lecture Notes in Computer Science, vol. 3855. Springer, Berlin Heidelberg, pp. 348–363.

O'Callahan, R.W., 2001. Generalized aliasing as a basis for program analysis tools. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, aAI3051029.

O'Callahan, R. W., Jackson, D., 1997. Lackwit: a program understanding tool based on type inference. In: Proceedings of the 19th International Conference on Software Engineering. ICSE'97. ACM, New York, NY, USA, pp. 338–348. http://doi.acm.org/10.1145/253228.253351.

Pearson, K., 1901. On lines and planes of closest fit to systems of points in space. Philosophical Magazine 2 (6), 559–572. http://stat.smmu.edu.cn/history/pearson1901.pdf.

R Development Core Team 2009. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org.

Rohn, J., Apr. 2010. Inverse interval matrix: A survey. Tech. Rep. V-1073, Academy of Sciences of the Czech Republic.

Rousson, V., Gasser, T., 2004. Simple component analysis. Journal of the Royal Statistical Society. Series C 53 (4), 539–555. http://dx.doi.org/10.1111/j.1467-9876.2004.05359.x.

Sankaranarayanan, S., Ivančić, F., Gupta, A., 2007. In: Nielson, H. R., Filé, G. (Eds.), Static Analysis, 14th International Symposium. SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007. In: Lecture Notes in Computer Science, vol. 4634. Springer, Berlin Heidelberg, pp. 366–383.

Sankaranarayanan, S., Sipma, H.B., Manna, Z., January 2005. Scalable analysis of linear systems using mathematical programming. In: Cousot, R. (Ed.), Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17–19, 2005. Proceedings. In: Lecture Notes in Computer Science, vol. 3385. Springer, Berlin Heidelberg, pp. 25–41.

Simon, A., King, A., Howe, J. M., 2003. Two variables per linear inequality as an abstract domain. In: Leuschel, M. (Ed.), Logic Based Program Synthesis and Transformation 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17–20, 2002. Revised Selected Papers. In: Lecture Notes in Computer Science, vol. 2664. Springer, Berlin Heidelberg, pp. 71–89.

Stursberg, O., Krogh, B.H., 2003. Efficient representation and computation of reachable sets for hybrid systems. In: Maler, O., Pnueli, A. (Eds.), Hybrid Systems: Computation and Control. 6th InternationalWorkshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003. Proceedings. In: Lecture Notes in Computer Science, vol. 2623. Springer, Berlin Heidelberg, pp. 482–497.

The Open University, 1983. MDST242 Statistics in Society, Unit C3: Is my child normal? The Open University, figure 3.12.

Tillmann, N., de Halleux, J., 2008. Pexwhite box test generation for.net. In: Beckert, B., Hahnle, R. (Eds.), Tests and Proofs. In: Lecture Notes in Computer Science, vol. 4966. Springer, pp. 134–153.